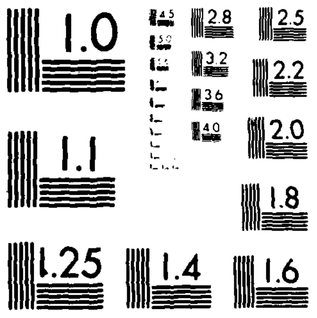
AD-A111 578 FORD AEROSPACE AND COMMUNICATIONS CORP PALO ALTO CA W--ETC F/G 9/2
KSOS COMPUTER PROGRAM DEVELOPMENT SPECIFICATIONS (TYPE B-5). (K--ETC(U)
DEC 80 MDA903-77-C-0333
UNCLASSIFIED WDL-TR7932-REV-3 NL

1 OF 4
AD
A111578

1115



2

SECURE MINICOMPUTER OPERATING SYSTEM (KSOS)

COMPUTER PROGRAM DEVELOPMENT SPECIFICATIONS (TYPE B-5)

Department of Defense Kernelized Secure Operating System

I. SECURITY KERNEL (CDRL 0002AF)
II. UNIX EMULATOR (CDRL 0002AG)
III. SECURITY-RELATED SOFTWARE (CDRL 0002AH)

Contract MDA 903-77-C-0333

Prepared for:

Defense Supply Service—Washington
Room 1D245, The Pentagon
Washington, D.C. 20310

Ford Aerospace &
Communications Corporation
Western Development
Laboratories Division

3939 Fabian Way
Palo Alto, California 94303

82

WDL-TR7932
Revision 3
December 1980

SECURE MINICOMPUTER OPERATING SYSTEM (KSOS)

SECURITY KERNEL COMPUTER PROGRAM DEVELOPMENT SPECIFICATION (TYPE B5)

Department of Defense Kernelized Secure Operating System

Contract MDA 903-77-C-0333
CDRL 0002AF

Prepared for:

Defense Supply Service - Washington
Room 1D245, The Pentagon
Washington, DC 20310

Ford Aerospace & Communications Corporation
Western Development
Laboratories Division

3939 Fabian Way
Palo Alto, California 94303

NOTICE

The Department of Defense Kernelized Secure Operating System (KSOS) is being produced under contract to the U.S. Government. KSOS is intended to be compatible with the Western Electric Company's UNIX*tm Operating System (a proprietary product). KSOS is not part of the UNIX license software and use of KSOS is independent of any UNIX license agreement. Use of KSOS does not authorize use of UNIX in the absence of an appropriate licensing agreement.

This document is a major revision of an earlier draft dated July 1978. That and other previous editions are obsolete and should not be used.

TABLE OF CONTENTS

Kernelized Secure Operating System (KSOS): Security Kernel Computer Program Development Specification (Type B5)

1. SCOPE

1.1 Identification

This specification establishes the performance, design, development and test requirements for the Kernelized Secure Operating System (referred to as "KSOS") Security Kernel. The KSOS Security Kernel is intended to support the emulation of the UNIX operating system while providing an environment suitable for the construction of new secure systems.

1.2 Functional Summary

The primary purpose of this specification is to define the functional requirements for a Security Kernel to support the emulation of the standard UNIX operating system. This specification further defines the design, test, performance, and quality assurance provisions for the KSOS Security Kernel.

This specification is intended to serve an additional purpose, that of defining a particular implementation of the KSOS Security Kernel on a Digital Equipment Corporation PDP-11/70 computer system. Thus, there are a number of PDP-11/70 specific points throughout the specification. For implementations of the KSOS Kernel on other hardware bases, these points should be treated as non-binding, informative commentary.

The KSOS Security Kernel shall support the emulation of the standard UNIX operating system within the constraints of the multilevel security model. The Security Kernel shall provide all the data and computational objects required to construct a general purpose operating system. The Security Kernel shall mediate all information exchanges within the system permitting only those exchanges which are consistent with the multilevel security model. The Security Kernel shall provide secure information channels which permit the user to communicate directly with "trusted" software processes. The secure information channels shall not be subject to compromise due to "spoofing" by untrusted software.

This specification is organized in the following way. First the relevant Kernel requirements are detailed. The presentation of the requirements includes the general interface requirements, interface block diagram, and the detailed interface definition. The detailed interface definition specifies the Security Kernel call interface and significant Kernel-related software elements, i.e., the scheduler and secure terminal interface. System adaptability is discussed followed by the specification of quality assurance provisions for the Security Kernel.

1.3 Terminology

The language used throughout this specification attempts to conform to the guidelines of Section 3.2.3 of MIL-STD-490. In particular, the word "shall" means that the specification expresses a provision that is binding. The words "should" and "may" mean that the specification expresses a provision which is non-mandatory. The word "will" is used to express a declaration of purpose on the part of the Government.

KSOS shall support the mandatory DoD security policy of DoD Directive 5200.1-R that is embodied in a Government approved mathematical model. [Verif 78] Proofs of the system's security properties shall be in terms of this model. KSOS shall provide a minimum of eight (8) hierarchical security classification categories (or simply security categories) and a minimum of thirty-two (32) mutually independent security compartments. The security categories shall be such that

UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP SECRET

Where "<" is defined in accordance with the requirements of DoD 5200.1-R. One security compartment shall be reserved for read protection of system data bases and programs. This compartment shall be called the "system" compartment. KSOS shall provide for Kernel-enforced integrity. Integrity is defined as the formal mathematical dual of security [Biba 75]. At least four (4) hierarchical integrity classification categories (or simply integrity categories) shall be provided in KSOS. The integrity categories include at least the following three classification categories:

USER < OPERATOR < ADMINISTRATOR

Although there is no immediate requirement for integrity compartments, the development contractor should include provisions to ease their later inclusion. For the remainder of this specification, the term "security level" means the combination of a security category and a set of security compartments (which may be null). The term "integrity level" means the combination of an integrity category, and a (presently always null) set of integrity compartments. The term "level" (or "access level") means the security and integrity level, that is, the combination of a security category, a set of security compartments (which may be null), an integrity category, and a (presently always null) set of integrity compartments. The "access matrix" defines for a particular user or group the combination of the maximum security level permitted for that user or group, all security compartments to which that user or group has access, the maximum integrity level for that user or group, and all integrity compartments (presently always null) to which that user or group has access.

In a secure system it is necessary to have processes external to the kernel which perform security critical functions. These have come to be called "trusted processes". In KSOS there is a refinement in this terminology. As Figure 1-1 shows there are subdivisions of the software inside the security perimeter. Some of the processes are "privileged". This means that they may violate one or more of the security rules enforced by the Kernel. These processes must be designed and implemented with the same care and thoroughness as the Kernel itself. Section 3.1.1.2.5 presents the privileges which may be

```
                    |-----------------------------|
                    |                             |
                    |    User process part        |   User domain
                    |                             |
                    |----------|------------      |
                    |          V           |      |
                    | Emulator process part |     |   Supervisor domain
                    |                      |      |
                    |----------|---------------|--|
|----------|        |          V               V  |
|          |        |     Security kernel         |   Kernel domain
| User     |        |  ^                          |
|          |        |--|--------------------------|
| Console  |-------------                         |
|          |        |  Host (PDP-11/70) computer  |
|----------|        |-----------------------------|
```

Figure 1-1. Terminology Figure

afforded processes. A second subdivision has been termed "responsible" processes. These are processes are not privileged to violate any of the Kernel's rules, but which are nonetheless important to the overall security of the system. Examples of responsible processes include the Secure Mail facility and the processes which manipulate the data bases describing the users' security and integrity permissions (the "access matrix"). These processes must also be carefully designed and implemented, but do not require formal specification or verification. Because they do perform security-related functions, they cannot include untrusted components (e.g. the UNIX Emulator). The third subdivision is the "encapsulated utilities" (EU in the figure). These are self contained functions to which the user simply supplies parameters. The encapsulated utilities are not privileged to violate Kernel rules, and do not affect the security of users. The chief function in the encapsulated utilities is system generation.

2. APPLICABLE DOCUMENTS

The following documents, of exact issue shown, form a part of this specification to the extent specified herein. In the event of a conflict between the referenced documents and the contents of this specification, this specification shall be considered a superseding requirement. In the text references to these documents are in the form [Name date], e.g. [Biba 75].

2.1 Government Documents

2.1.1 Directives, Manuals and Standards

a. DoD 5200.1-R Information Security Program Regulation

b. DoD 5200.28 Security Requirements for Data Processing (ADP)

c. DoD 5200.28-M ADP Security Manual

d. MIL-STD-483 Configuration Management

e. MIL-STD-490 Specification Practices

f. MIL-STD-1521A Technical Reviews and Audits

2.1.2 Reports

a. [A-Specs 78] "KSOS System Specification (Type A)", WDL-TR7808 Revision 1, Ford Aerospace and Communications Corporation, Palo Alto, CA (July 1978).

b. [Bell and LaPadula 73] Bell, D.E. and LaPadula, L.J., "Secure Computer Systems", ESD-TR-73-278, Volume I-III, MITRE Corporation, Bedford, MA (November 1973 - June 1974).

c. [Biba 75] Biba, K.J., "Integrity Considerations for Secure Computer Systems", MTR-3153, MITRE Corporation, Bedford, MA (June 1975).

d. [Emulator 78] "KSOS Computer Program Development Specification (Type B5): UNIX Emulator", WDL-TR7933, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

e. [NKSR 78] "KSOS Computer Program Development Specification (Type B5): Non-Kernel Security-Related Software", WDL-TR7934, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

f. [Verif 78] "KSOS Verification Plan", WDL-TR7809, Ford Aerospace and Communications Corporation, Palo Alto, CA (March 1978).

g. [Walter et al. 74] Walter, K.G. et al., "Primitive Models for Computer Security", ESD-TR-74-117, Case Western Reserve University, Cleveland, OH (January 1974).

2.2 Non-Government Documents

a. [BBN 75] "Interface Message Processor, Specifications for the Interconnection of a Host and an IMP", Report 1822, Bolt Beranek and Newman Inc., Cambridge, MA, (December 1975).

b. [Holmgren et al. 77] Holmgren, S.F., Healy, D.C., Jones, P.B. and Kasprzycki, E., "Illinois Inter-Process Communication Facility for UNIX", CAC Technical Memorandum 84, CCTC-WAD Document 7507, Center for Advanced Computation, University of Illinois at Urbana-Champaign, Urbana, IL (April 1977).

c. [Lampson 73] Lampson, B., "A Note on the Confinement Problem", CACM, Volume 16, Number 10, pp 613 - 615 (October 73).

d. [Lipner 75] Lipner, S.B., "A Comment on the Confinement Problem", Proc. Fifth Symposium on Operating Systems Principles, ACM SIGOPS Review, Volume 9, Number 5, pp 192 - 196 (19-21 November 1975).

e. [Nemeth et al. 77] Nemeth, A.G., Sunshine, C., Zucker, S. and Tepper, S., "Progress Towards a DeFacto DoD UNIX Inter-Process Communication Standard", Working Paper, (May 1977). (Available from the authors.)

f. [Parnas 72] Parnas, D.L., "A Technique for Software Module Specification with Examples", CACM, Volume 15, Number 5, pp 330 - 336 (May 1972).

g. [Postel 78a] Postel, J.B., "Internetwork Protocol Specification", Version 4, Information Sciences Institute, University of Southern California, Marina del Rey, CA (February 1979).

h. [Postel 78b] Postel, J.B., "Specification of Internetwork Transmission Control Protocol - TCP", Version 4, Information Sciences Institute, University of Southern California, Marina del Rey, CA (February 1979).

i. [Roubine and Robinson 77] Roubine,O., L.Robinson, Special Reference Manual, 3rd ed., Technical Report CSG-45, SRI International, Menlo Park, CA (January 1977).

j. [Sunshine 77] Sunshine, C., "Interprocess Communication Extensions for the UNIX Operating System: I Design Considerations", R-2064/1-AF, RAND Corporation,

k. [Zucker 77] Zucker, S., "Interprocess Communication Extensions for the UNIX Operating System: II Implementation", R-2064/2-AF, RAND Corporation, Santa Monica, CA (June 1977).

2.3 Other References Not Part of This Specification

a. [Thompson 75] Thompson, K. and Ritchie, D.M., "UNIX Programmer's Manual", Sixth Edition, Western Electric Corporation, Greensboro, NC (May 1975).

b. [Ritchie 74] Ritchie, D.M. and Thompson, K., "The UNIX Timesharing System", CACM, Volume 17, Number 5, pp 365 - 375 (May 1974).

c. [UNIX 75] "UNIX Program Listings", Version 6.0, Western Electric Corporation, Greensboro, NC (May 1975).

3. REQUIREMENTS

3.1 Computer Program Definition

This Computer Program Configuration Item (CPCI) defines a provably secure operating system kernel which enforces DoD security regulations upon computer computations. The initial implementation of the Security Kernel for KSOS shall execute upon a Digital Equipment Corporation PDP-11/70 computer system. The minimum, typical and maximum hardware configurations to be supported by KSOS are described in the System Specification [A-Specs 78].

3.1.1 Interface Requirements

The KSOS Security Kernel shall have the following interfaces between major system components:

a. between all software elements and the host computer hardware;

b. between all software executing in the supervisor and user virtual memory domains of the PDP-11/70;

c. between the Kernel and all privileged security related subsystems defined in the Non-Kernel Security Related Software CPCI [NKSR 78].

d. between the user terminal and the Security Kernel.

All information exchanges between user programs, privileged subsystems, files, and devices shall be mediated by the KSOS Security Kernel.

3.1.1.1 Interface Block Diagram

Figure 3-1 is the interface block diagram for the KSOS Security Kernel. This diagram illustrates the layers of virtual machines or functional capability of KSOS. The Security Kernel interfaces directly with the KSOS UNIX Emulator CPCI, user programs, and the host (PDP-11/70) computer. Both the Emulator and the user programs shall communicate directly with the Kernel through an inter-domain call instruction such as the PDP-11 EMT instruction. To preserve compatibility with existing UNIX applications on the PDP-11/70, the PDP-11 TRAP and IOT instructions shall be reserved for the emulation of the UNIX operating system. The Security Kernel shall support secure communication paths from the user terminal to the trusted system software.

Figure 3-1. Interface Diagram

3.1.1.2 Detailed Interface Definition

This section further details the interfaces between the KSOS Security Kernel and the software elements listed in Section 3.1.1, Interface Requirements. The objects supported by the Kernel are discussed, followed by a presentation of the structure of processes executing on the Security Kernel. Finally, these concepts are applied to the detailed interface specification between the KSOS Security Kernel and the other KSOS software elements.

3.1.1.2.1 Kernel Objects

Five objects shall be supported at the Kernel interface: processes, process segments, files, devices and subtypes. The next section discusses the five Kernel objects and the mechanism used to name the objects and to retain control information concerning the object. This discussion is followed by a presentation of the Kernel primitive operations upon these objects.

All Kernel objects shall be referenced by a common designator called the Secure Entity Identifier (SEID, pronounced "seed"). The SEID shall be separated into two parts, the name space partition and the name within the partition. Only one object type shall be assigned to each partition. However, more than one partition may have the same type. The name space partition shall be used to distinguish the object type: file, device, process segment, file subtype, and process. The name space partition for a file designates the logical volume ("extent") on which the file resides. The System Administrator should make sure that the logical volumes that could be mounted on his system(s) have unique name space partitions. The Immigration Officer function of the Non-Kernel Security Related Software [NKSR 78] aids in this task by maintaining a data base of name space partitions in use. The name within the partition shall be used to specify particular instances of an object within the partition. A SEID shall be returned as the result of new object creations (i.e. K_create, K_build_segment, K_fork, and K_spawn.)

Associated with each Kernel object are two classes of information:

a. type independent information: information common to all objects. The type independent information includes:

   1. the security level of the object (classification category and compartment set)

   2. the integrity level of the object (classification category and compartment set)

   3. the discretionary access control information. As in UNIX, this information shall consist of three sets of permissions:

      - for processes with the same user ID as the object (in UNIX "the owner"),

      - for processes with different user IDs, but the same group ID ("the group"), and

      - for processes with both different user IDs and group IDs ("all others").

   Each permission shall consist of three independent access modes:

      - read,

      - write, and

- execute/search.

4. the object's owner (a user ID and a group ID)

5. the set of privileges associated with the object.

b. the type dependent information: information whose content varies by the type of the object

The type independent information for an object contains all of the information required by the Kernel to make security decisions (i.e. to permit or deny an access attempt by a process to an object). Because the type independent information is the same for all objects, the security checking may be implemented more simply. As is shown in the formal specifications for the Kernel (Appendix A to this specification), all mandatory access checking can be localized in a single function, SMXflow, which checks whether information may legitimately flow from the requested source to the requested destination.

The discretionary access checking can be similarly localized in a single function, SMXdap. The Kernel shall normally enforce only the read and write permissions. However, some processes possess the privilege to use the execute/search permissions for read permission ("realize execute permissions"). This privilege is used to faithfully emulate UNIX execute-only files (by the Process Bootstrapper CPC) or search-only directories (by the Directory Manager CPC).

3.1.1.2.1.1 Kernel Object: Processes

In KSOS the process is the only active agent in the system. All the other Kernel-supported objects are passive. The type independent information for the process and for the object being accessed shall be used by the Kernel to determine whether to permit or deny access attempts by the process. A process shall have two owner identifications: the effective owner (user ID and group ID) and the real owner. The effective owner shall be used in discretionary access checking, and shall be inherited by new objects created by this process. Thus the effective owner is part of the type independent information. The real owner is part of the type dependent information for the process.

Because a process is itself an object, the Kernel shall enforce the mandatory and discretionary access policies on accesses to the process. In particular, interprocess communication (IPC) shall only be permitted if information may flow from the sending process to the recipient.

The type dependent information for a process shall include:

a. the identities (SEIDs) of this process and its parent process (the parent process is the process which brought this one into existence via a K_fork or a K_spawn Kernel call)

b. the process family name

c. the real owner (user ID and group ID)

d. the program counter (pc) (PDP-11 specific)

e. the program status word (ps) (PDP-11 specific)

f. the pseudo interrupt level

g. the advisory priority

h. the timer alarm value (causes a pseudo interrupt when it counts down to zero)

i. the supervisor mode execution time (PDP-11 specific)

j. the user mode execution time

k. the timer toggle TRUE => pseudo interrupt every clock tick FALSE => no interrupt

In order to provide an operationally viable system, the Kernel shall provide support for privileged processes. Privileged processes shall be either created at system initialization or become privileged through the execution of the Kernel calls K_invoke or K_spawn specifying that a privileged instruction segment is to replace the segments of the process. The privileges of a process shall persist until another K_invoke is issued by the process, at which time the new privileges are established (K_invoke can also remove any special privileges for the process) or until a suitably privileged process explicitly alters them through a K_set_object_level call. The privileges that may be granted to a process are described in 3.1.1.2.5.

The UNIX Emulator [Emulator 78] (with the exception of the Directory Manager) and all UNIX user domain programs shall have no special privileges. Thus, the Emulator may be modified by KSOS sites to provide locally desired UNIX calls. Naturally, care must be taken in such modifications so that the remainder of the UNIX interface provided by the Emulator is unaffected. However, such changes cannot affect the security properties of the system, because the Emulator is not "trusted" by the Kernel, and cannot violate any of the security rules of the system.

All privileged software shall communicate directly with the Kernel. That is, such software shall use the Kernel calls directly and shall not use the (untrusted) UNIX Emulator. If privileged software is modified, it must be subject to the same rigor (e.g. formal specification, etc.) that were employed in its initial creation. Thus, most KSOS sites will not be allowed to modify or create privileged software.

To the KSOS Kernel, a process shall occupy multiple, separate virtual memory domains. In the PDP-11/70 KSOS implementation, a process shall be composed of three parts:

a. User domain process segments

b. Supervisor domain process segments

c. Processor state (registers, etc.)

These Kernel objects shall be visible to programs operating within the supervisor and user domains. The process may also request the Kernel to set or get its type independent and type dependent information via the appropriate Kernel calls.

3.1.1.2.1.2 Kernel Object: Process Segments

A process segment shall be a portion of the virtual memory of a process. A process shall be capable of having a subset of its active segments actually mapped into its virtual address spaces. Process segments shall be variable sized. A single segment shall be capable of spanning a virtual address range from a hardware limited lower bound up to the entire virtual address space in a domain. The development contractor may elect to provide segments in only a pre-defined set of sizes. All process segments, regardless of domain, shall be manipulated by the same set of Kernel segment primitives. User domain segments shall be directly addressable to the Emulator (e.g. through the PDP-11/70 MFPI(D) instructions) in order to transfer operating system call arguments. Programs executing in the user domain shall not directly observe or modify the supervisor domain. User and supervisor programs shall not directly observe or modify Kernel memory. In the general, machine independent case, domains of higher privilege shall be able to observe and to modify domains of lesser privilege. Domains of lesser privilege shall not have any access rights to domains of greater privilege.

Like the other Kernel objects, each segment shall have type independent information associated with it. This information is used by the Kernel to mediate access attempts to the segment. Specifically, the Kernel shall use the type independent information in determining whether other processes may map the segment into their virtual address space and in setting up the hardware memory mapping/protection registers for the processes permitted access to the segment. The privileges contained in the segment's type independent information shall be used by the K_invoke and K_spawn mechanisms. The process privileges shall be set from the privileges of the intermediary specified in the call. The type dependent information for segments shall include the following:

a. Sharable | Non-Sharable (This information supplements the discretionary access information. A segment might be marked as readable and writable by its owner, but still not be sharable to other processes with the same owner, for example the data segments of a process.)

b. Swappable | Locked in physical memory.

c. Status when the reference count drops to 0 (destroy segment|save segment) (the UNIX "sticky bit", ISVTXT)

d. Direction of segment growth (upward | downward)

e. Advisory position for the segment when primary memory resident

f. Executable | Not Executable (This is more generic. In the PDP-11/70 case, the segments placed in data space may be executable also.)

g. the direction of growth for the segment (Up | Down)

h. the segment size

This information and the type independent information for the segment shall be used by the Kernel in determining whether to permit or deny attempts by processes to access the segment (map it into their address space). For permitted accesses, this information shall be part of the information used to set up hardware protection and/or memory management registers for processes mapping the segment into one of their virtual address domains. For each process mapping the segment into one of its virtual address domains, that use of the segment shall have associated with it:

a. the domain into which the segment may be mapped

b. the virtual address in that domain at which the segment starts

c. whether the segment resides in the Instruction or Data spaces (for the PDP-11/70 implementation.)

d. whether the segment is presently active (mapped into the virtual address space)

These attributes shall control the use of the physical memory allocated to the segment. This information may be returned with the other type dependent information about the segment.

The concept of KSOS process segments is intended to be machine independent. No assumptions are made about the allocation or use of the native memory management hardware. Only two effects of the underlying hardware shall be visible to the Kernel user:

a. When a segment is allocated, a new mapping register(s) shall be allocated in a manner such that all segments shall be individually mapped and unmapped from the physical segmentation hardware.

b. Attempts to map a segment into a process virtual address domain shall fail when the pool of available memory management resources (i.e., registers) or virtual memory space has been exhausted or when there would be overlap of the virtual memory spaces of two segments.

The intent of this approach is to maintain a high degree of machine independence.

In the PDP-11/70 implementation of KSOS as many as sixteen memory management registers shall be available for segment mapping per process domain memory space. If the Instruction and Data (I and D) space separation feature of the PDP-11/70 hardware is not utilized, only eight registers shall be available to a process within a domain. The maximum length of a memory vector supported by a PDP-11/70 map register is 4096 words. In order to create process segments that are longer than 4096 words, more than one memory management register shall be used. (It should be recognized that in order to span the maximum virtual memory of the PDP-11/70, segments must be an integral multiple of 4096 words and must be aligned on 4096 word boundaries.) The ultimate size of a KSOS process will be limited by the addressing capability of the PDP-11/70; 64K words if I and D separation is enabled, 32K words if it is disabled. Standard UNIX process segments can be emulated by the following combinations of attributes:

a. Program pure text:

   1. Separated (in UNIX, a "411 magic' number"): Sharable, swappable, upward growing, instruction space placement, executable, read only

   2. Non-separated (in UNIX a "410"): Sharable, swappable, upward growing, data space placement, executable, read only

b. Data, BSS and impure text: Non-Sharable, swappable, upward growing, data space placement, Non-executable, read/write

c. Stack: Non-Sharable, swappable, downward growing, data space placement, Non-Executable, read/write (In addition, UNIX conventionally starts the stack at the highest virtual address and expands it downward.)

d. IPC segments: Sharable, swappable, upward growing, data space placement, Non-Executable, read/write (or read-only)

Other combinations would be possible. For example, it may be possible to implement multiple shared text segments, permitting the construction of the common subroutine mechanism as suggested in the proposals for UNIX IPC [Holmgren et al. 77] [Nemeth et al. 77] [Sunshine 77] [Zucker 77]. Existing UNIX programs which do not use shared IPC segments shall find the KSOS process segment limitations identical to those of standard UNIX.

3.1.1.2.1.3 Kernel Object: Files

The file system visible at the Kernel interface shall be a "flat" file system. No structure shall be imposed upon the organization of files. File directories will be supported by the UNIX Emulator and the Directory Manager. Hence, the file objects at the Kernel level shall be homogeneous. The Kernel files may be thought of as linear sequences of data blocks. The Kernel shall place no special interpretation on the contents or format of the files, e.g. there is no Kernel support for more sophisticated access methods.

The type dependent information associated with a file shall include:

a. the size of the file

b. the link count for the file

c. the time of the last modification to the file

d. the subtype (if any) to which the file belongs

e. whether or not the file is open for writing (or was open when the system halted)

A process shall be able to access the file data contents only if the multilevel security/integrity conditions are satisfied and both the file subtype and file discretionary permissions given to the effective group and user ID of process properly match the requested file operation.

Files shall be referenced at the Kernel level through the use of the Secure Entity Identifier (SEID) that uniquely identifies the file. Access to a file shall be granted by the Kernel before any input or output may be performed to the file. The operation of opening a file shall return an open object descriptor in a manner similar to the current Standard UNIX file open. The open object descriptor shall be used to reference the file in all subsequent Kernel calls. (The operation of opening a file may be viewed as being equivalent to granting a capability to the data object in a capability based Kernel.)

3.1.1.2.1.4 Kernel Object: Devices

As in UNIX, in KSOS devices may be thought of as a special class of files. The type dependent information for devices shall be include the same information as the type independent information for files. The type dependent information for devices may include additional information used internally by the Kernel. Devices SEIDs shall be permanently assigned in the Kernel. However, any UNIX directory entry may point to a device. Because the underlying devices upon which the file system is built are visible at the Kernel interface (i.e. they may be opened by suitably authorized processes), the System Administrator should assure that these devices are protected from unauthorized modification or access by means of appropriate discretionary access and subtype membership. The Kernel shall not allow files to be created whose security level is greater than the underlying device on which they are being created.

3.1.1.2.1.5 Kernel Object: File Subtypes

File subtypes are a type extension feature. The goals of the KSOS Security Kernel file subtype mechanism are to:

a. Subdivide the Kernel file object type

b. Provide a separate subtype manager for each file subtype controlling the use of specially typed files (e.g. directories)

c. Localize subtype management improving leverage on the proof process

File subtypes, because they are Kernel objects, shall have both a Secure Entity Identifier (SEID) and a type independent information. Thus, file subtypes shall have a security and integrity level associated with each instance of a subtype. Most importantly, file subtypes shall have an owner and a discretionary permission list. The subtype owner shall refer to the group and user ID of the subtype manager. The discretionary permission list shall describe the permitted access modes to all files of that subtype by the subtype manager (owner), group, and all users.

The nature and structure of file subtypes is best revealed by examining their use in the Kernel level file open function, K_open. In order to open a file for access in a specific mode, the process shall successfully meet four classes of conditions:

a. Mandatory security and integrity

b. Subtype capability or file subtype discretionary permission

c. File data discretionary permission

The subtype capability of a process may be either implicit or explicit. Implicit subtype capability shall be determined by the effective group and user ID of the process. If the requested access mode successfully matches the permissions accorded to the effective group and user ID of the process for that file subtype, the file shall be opened. Explicit subtype capability shall be acquired by performing a K_open call on the subtype, specifying the subtype by its SEID. Subsequent K_open calls applied to files belonging to this subtype shall accept the open object descriptor returned by the subtype open operation as an explicit capability (permission) to manipulate the file in the requested mode.

As an example of the use of file subtypes, the Emulator uses the Kernel subtype mechanism to implement UNIX directories. Providing that they meet the mandatory and discretionary access checks, ordinary users are permitted read access to directory files, but are denied write access. Only the Directory Manager is permitted to have write access to directory subtype files. Hence, all user processes are able to search directories and perform their own path name interpretation. However, only the Directory Manager is able to update or delete directories, insuring the integrity and accuracy of directory files.

3.1.1.2.2 Process Structure

The logical information which determines a specific instance of a process, in the PDP-11/70 implementation of KSOS, shall span three memory domains. The data contained at each domain level is characterized by the function it is intended to serve:

a. User domain: User process information (i.e., program instructions, data, stack, etc.)

b. Supervisor domain: Information required to faithfully emulate the UNIX operating system. Further, all Non-Kernel Security-Related Software functions which operate with special privileges or which manipulate security critical data bases shall utilize only Kernel calls (i.e. these functions shall not utilize the Emulator) and shall reside in the Supervisor domain.

c. Kernel: Process data needed to multiplex the hardware, maintain the logical determinacy of user processes, and to enforce multilevel security.

Unprivileged processes executing in either user or supervisor domain shall be regarded as "untrusted" by the Kernel domain software. That is, the security, integrity, and discretionary access policies shall be enforced by the Kernel for all unprivileged processes.

3.1.1.2.3 Hardware Interface

The KSOS Security Kernel shall execute correctly on all hardware configurations specified in the System Specification (Type A) [A-Specs 78].

The following general remarks can be stated about the KSOS hardware requirements. The KSOS design shall execute on a host computer with a minimum of three separate virtual memory and privilege domains. The KSOS Security Kernel shall reside in the most privileged host processor domains. Non-Kernel software shall reside in processor domains where all accesses to Kernel-supported system resources can be mediated. This means that non-Kernel software may not change its virtual to physical address mapping, potentially gaining access to information resident in other processes, files, or device registers. The processor shall also provide an interdomain call mechanism which permits services to be performed on its behalf by software executing in another virtual memory and privilege domain. That is, the processor shall support some sort of "trap" mechanism for requesting services from more privileged software.

3.1.1.2.4 Emulator/User Mode Software Interface

In addition to the mediation of information accesses, the purpose of the KSOS Security Kernel is to provide a service environment to support applications software executing in Kernel-controlled processes. This specific implementation effort requires the emulation of the Western Electric Corp. UNIX operating system interface as defined in the System Specification. All non-Kernel software shall have a direct call interface to the Security Kernel. Hence, any non-Kernel software shall be able to directly utilize Kernel level services. Briefly, these Kernel level services are the manipulation of system resources, i.e. files, devices, processes, process segments, and file subtypes. Section 3.2 of this document details the Security Kernel functions available through the direct Kernel call interface. Applications programs may include both UNIX calls and Kernel calls. However, the application program designer must take care that the use of both types of calls does not lead to incorrect results due to interactions between the Emulator's use of Kernel calls, and that of the application program. In no cases can such incorrect interaction lead to security, integrity, or discretionary access violations. Rather, the potential problem is in incorrect results or garbling of files.

In order to accurately emulate the UNIX operating system interface on the PDP-11/70 host processor, the TRAP instruction shall be used to transfer user domain requests to the UNIX Emulator operating in the supervisor domain of the machine. The IOT instruction shall also cause a trap to the Emulator for the invocation of core dumps by the Emulator at the request of the user domain program. The EMT instruction shall be reserved for direct user domain to Kernel and supervisor domain to Kernel interdomain calls. Parameters to Security Kernel calls should (in the PDP-11 implementation) be passed on the memory stack of the calling domain. To facilitate the transfer of lengthy blocks of data (i.e. K_device_function, K_read_block, and K_write_block) an address pointer (called a "parameter block") to the data area may be used as a valid parameter. The Kernel should use this address to fetch/store information from the data area. Return values from the Security Kernel should be transfered to the caller in a register (i.e. register R0) where possible. When the Kernel is requested to transfer data into or out of the user's memory, the data to be transfered must be completely contained within one process segment. This shall include the parameters passed in or returned on the stack.

The KSOS Security Kernel shall reflect certain processor traps to the Supervisor domain process, i.e., the UNIX Emulator. The processor exception conditions reflected outwards shall be those traps caused by the software executing outside the domain of the Kernel. The PDP-11/70 traps to be reflected outward by the Kernel are:

a. Illegal or reserved instruction exceptions.

b. Breakpoint traps.

c. IOT instruction traps. (Causes UNIX abort.)

d. TRAP instruction. (Causes entry to UNIX emulator dispatch.)

e. Floating point errors.

f. Memory management traps.

The means through which these traps shall be communicated to the executing process shall be a pseudo interrupt. The pseudo interrupt may include additional information, such as the cause of the trap, or any relevant processor status required by the extra-Kernel software to process the trap.

3.1.1.2.5 Privileged Subsystem Interfaces

To provide a useful, operating environment, some non-Kernel software must be privileged (trusted) to perform system maintenance and other functions that selectively violate the protection policy of the Kernel. The KSOS Security Kernel shall maintain the process privilege information which determines the policies that the process has been permitted to violate. Privileged software shall be brought into execution either at system initialization time or through the execution of the K_invoke or K_spawn Security Kernel primitives. An orderly system initialization procedure shall be implemented which initializes the trusted processes required for correct operation of the overall system. The K_invoke (or K_spawn) primitive shall derive the process privileges from the designated instruction segment in a manner which guarantees the execution of that segment with the privilege set. One particular instruction segment, containing the Process Bootstrapper CPC, shall have the privilege to set privileges, and shall be a mechanism by which a process begins executing a potentially priviliged image contained in a file. The Process Bootstrapper shall set the privileges of the process to those of the file from which it was initialized. The privilege information associated with the process image file is normally set only by the Privilege Control Editor which may only be executed by the System Administrator or System Security Officer while running at the ADMINISTRATOR integrity level. (See the Non-Kernel Security Related Software B5 Specification [NKSR 78].)

The following special privileges shall be capable of being granted or revoked:

a. The ability to violate the simple security property. (This privilege is not presently used. It is included for completeness.)

b. The ability to violate the simple integrity property.

c. The ability to violate the *-property for security.

d. The ability to violate the *-property for integrity.

e. The ability to violate the security compartment checking rules.

f. The ability to violate discretionary access. (This privilege is not used. It is included for completeness.)

g. The ability to use the execute discretionary access permissions for read access. (This is also called "realize execute permissions".)

h. The ability to use the K_secure_terminal_lock primitive.

i. The ability to use the K_link and K_unlink primitives.

j. The ability to use the K_mount and K_unmount primitives.

k. The ability to change the following type independent information about an object:

   1. The security level of the object.

   2. The integrity level of the object.

   3. The the owner (user ID and group ID) of the object.

l. The ability to change the privileges in the type independent information for the object.

m. The ability to set type dependent information (status) about an file (i.e., the ability to execute K_set_file_status).

n. The ability to save a segment in the swapping area (i.e. the ISVTXT or "sticky bit").

o. The ability to lock a segment in core (not swappable).

p. The ability to use the K_signal primitive.

q. The ability to use the K_walk_process_table primitive.

r. The ability to use the K_halt primitive.

s. The ability to use the Kernel primitives from user mode. (This privilege is included for protection of the Emulator. It shall be possible to disable it, allowing unrestricted use of Kernel calls from user domain.)

It should be noted that the owner of an object may always change the discretionary access information for that object, hence no special privilege is needed.

There are some privileges which subsume others. For example, the ability to set the level of an object (change its type independent information) is functionally equivalent to being able to violate the simple security, security *-property, simple integrity, integrity *-property, and discretionary access rules. In general privileged processes should be afforded the minimum privileges that allow them to perform their intended functions. Programs which need the more powerful privileges, e.g. set object level, or set privileges, should be subject to the most design and implementation rigor.

3.1.1.2.6 Secure Terminal Interface

Certain privileged processes such as the Secure Server (part of the Non-Kernel Security-Related Software CPCI [NKSR 78]) require a secure communication path from the user terminal to transfer passwords and other information which cannot be contaminated or compromised. Further, the user must have the assurance that he is communicating solely with the Kernel or trusted subsystem and not a user process masquerading as a trusted program. The Secure Terminal Interface function of the KSOS Kernel shall provide this mechanism.

Two sets of terminal device SEID's shall be maintained, the user (thawed) set and the secure (frozen) set. The user set shall determine the access capabilities of regular user processes to specific terminals. The secure set shall determine the access capabilities of secure software to specific terminals. The secure set shall be protected by the discretionary access mechanism. Their owner shall be a distinguished user ID, and software allowed to use them shall execute in the discretionary access domain of this user ID. The terminal is said to be "frozen" when the secure path is active and the normal path is blocked. It is said to be "thawed" when the normal path is active. A terminal shall become frozen when the user enters the secure attention character, a character reserved for this one purpose. When the Kernel receives the secure attention character it shall freeze the terminal and send an IPC message to the Secure Initiator [NKSR 78]. A terminal shall become thawed only by having a privileged process issue a K_secure_terminal_lock Kernel call. When a terminal is frozen, all input and output on the normal path shall be blocked. It is the responsibility of the processes privileged to thaw terminals to assure that only the correct path is thawed, i.e. that the processes using that path have the appropriate security and integrity level to use the terminal at its (possibly new) level.

In the frozen mode, only software privileged to use the secure path may interact with the user. (The exact privileges of such software shall be the privilege to use K_secure_terminal_lock and the ability to execute in the discretionary access domain user ID which owns the secure terminal devices.) In order to communicate with a frozen terminal, a privileged process shall open the terminal using the corresponding terminal SEID selected from the secure terminal set. Only privileged software shall successfully open terminals using SEID's from the secure terminal set. The returned open object descriptor may then be used by the secure process to perform read and write operations to and from the terminal.

3.2 Detailed Functional Requirements

The functional descriptions in this section use the same argument names and order as the formal specifications found in Appendix A. However, these discriptions have expanded out various structures into their components for clarity and flexibility of implementation. Subject to Government approval, the development contractor may package the arguments as needed for ease of implementation and use.

Many of the following functions have a list of exception conditions which must be satisfied for the function to have any effect. The order of these exceptions in the text is not necessarily the order in which they should be checked in an implementation of the Kernel. Rather, the order of the exceptions in the formal specifications found in Appendix A shall be used. The development contractor may add additional exceptions, subject to Government approval. In some cases, the true cause of an exception cannot be returned to the user because it could be used as a confinement channel [Lampson 73] [Lipner 75]. In these cases, both the true cause and the returned value are noted.

Some of the functions utilize parameter blocks to designate areas in which to accept or return data. A parameter block is a structured argument that specifies an area of virtual memory for the process. For the PDP-11/70 implementation, a parameter block shall consist of:

a. the domain in which the data resides (user or supervisor)

b. the space in which the data resides (instruction or data space)

c. the starting virtual address of the data

d. the size of the data block (this may be omitted if a standard size is always used)

In all cases, an exception shall be raised if the data area specified by the parameter block does not lie completely within one segment which is accessible to the caller. That is, the segment containing the data specified by the parameter block must be capable of being accessed in the implied mode, For data that is being fetched, the segment must be readable, and for data being returned, the segment must be writable. Where no ambiguity results, the phrase "contained in the parameter block" (or similar) is used in lieu of the more precise "contained in the data area designated by the parameter block" in the interests of brevity.

3.2.1 Kernel Function: K build segment

3.2.1.1 Inputs

The Kernel primitive K_build_segment shall take the following parameters.

ss.gl.shareable: Boolean flag for sharing FALSE => segment is not sharable, TRUE => sharable
ss.gl.lock: Boolean flag for locking TRUE => do not swap (segment is locked in memory) (requires privilege to lock a segment) FALSE => swappable
ss.gl.sticky: Boolean flag for segment status when reference count goes to

zero (the "sticky bit" of UNIX) TRUE => do not destroy segment (requires privilege to set the sticky bit) FALSE => destroy segment (normal case)

ss.gl.memAdvise: Boolean flag designating advisory information to the Kernel regarding placement of the segment in memory, TRUE => segment will have I/O directed to it, FALSE => no I/O. This is merely advisory, I/O can be done to any segment accessible in the required mode.
ss.gl.executable: Boolean flag indicating whether this segment is executable
ss.gl.growth: the direction of segment growth (up or down)
ss.size: the initial segment size in bytes (the development contractor may round this up to the next integral unit of allocation)
ss.access: access allow calling process to the segment
ss.vl.domain: domain of allocation/mapping
ss.vl.idSpace: Instruction Space or Data Space placement (PDP-11/70 specific)
ss.vl.vAddr: virtual address of first byte in the segment
ms: the initial discretionary access permissions (read, write, execute/search for the owner, group, and all others)

3.2.1.2 Processing

K_build_segment shall create a new process segment and map it into the process. Memory management registers shall be used by the segment according to the address range specified for the segment. An error shall be returned if the address range specified for the segment would be in conflict with the address range of some other mapped in segment.

The discretionary access parameter shall determine the type of machine reference permitted to the segment and which other processes may share the segment if sharing is permitted. Accesses shall be selected from the set {read, write, execute}. The segment shall be initialized to contain all zeros. The process must store whatever data it requires into the segment explicitly. The owner of the segment shall be taken from the effective owner of the process. The security and integrity levels of the segment shall be taken from those of the process.

Segments may be shared between processes providing that the security, integrity and discretionary access checks would allow such sharing. Sharing of segments requires that the first process to desire to share the segment create it. Subsequent to creation, the segment SEID must be made available to processes wishing to share the segment, typically either via placing it in a mutually agreed upon file, or by passing it via an IPC message. The other processes would then issue K_rendezvous_segment calls (below) to gain access to the segment.

It is the responsibility of the processes sharing the segment to see to it that it is properly initialized, either by the Kernel's guarantee of all zeros, or by explicit initialization. The initialization may require cooperation or mutual exclusion to be completed successfully. This is particularly true in the case of shared pure text segments which are to be resident in the Instruction Space on the PDP-11/70. Since such segments cannot be written into, they must be created as writable segments, initialized from the appropriate image file, and then have their status changed to make them reside in the correct space and be non-writable. Care should be taken in the design of programs like the Process Bootstrapper which perform such initializations to assure that duplicate initialization attempts or multiple copies of the same shared text segment do not occur.

The virtual address and domain parameters shall be for the calling process only. Other processes sharing the segment may map it into their virtual address spaces as desired through their use of the K_rendezvous_segment calls. Of course, some segments, e.g. text segments, may operate correctly only if mapped starting at a specific virtual address.

3.2.1.3 Outputs

The Kernel primitive K_build_segment shall return the following information.

ec: error conditions
result.segSeid: the seid of the created segment
result.segd: designator name (process local) of newly created segment

The Kernel primitive K_build_segment shall detect and return the following exception (error) conditions.

EX: insufficient privilege, attempted to make new segment either sticky or locked in memory without the appropriate privilege
EX: bad discretionary access mode, attempted to create a segment which was writable but not readable for some permission (owner, group or other)
EX: bad size, segment size was too large
EX: global resource exhaustion (this is a potential channel)
EX: too many active segments
EX: virtual memory error, virtual memory address space would be exceeded or overlapped, or would fail to touch an 8K byte boundary

3.2.2 Kernel Function: K close

3.2.2.1 Inputs

The Kernel primitive K_close shall take the following parameters.

od: open object descriptor

3.2.2.2 Processing

The file referenced by the K_close call shall be inaccessible until the file has been successfully opened again. If the object associated with the open descriptor, od, is a file whose link count has been decremented to zero, and there are no other opens to it (i.e. no other open descriptors in the system are associated with it), then the file shall be deleted as part of the K_close processing.

3.2.2.3 Outputs

The Kernel primitive K_close shall return the following information.

ec: error condition

The Kernel primitive K_close shall detect and return the following exception (error) conditions.

EX: open object descriptor does not specify an open object

3.2.3 Kernel Function: K create

3.2.3.1 Inputs

The Kernel primitive K_create shall take the following parameters.

nspSeid: SEID from the name space in which to create file
ms: discretionary access permissions
stCap: subtype capability, an open descriptor for a subtype. The newly created file will belong to that subtype if the call is successful.

3.2.3.2 Processing

K_create shall be the mechanism by which new files are created. K_create differs from the UNIX creat(II) call in that K_create always creates a new file. K_create shall attempt to create the new file in the name space partition of the SEID supplied as an argument. The file control information shall be allocated and initialized. The Kernel level file name (SEID) shall be returned for potential inclusion in an Emulator level directory or similar construct. In addition, an open object descriptor shall be returned. This shall be equivalent to the user having requested a K_open for writing on the newly created file, with the exception that the discretionary access permissions are not checked. (This is to allow the common UNIX construct of lock files created with no access permissions.) The discretionary access shall be taken from the supplied argument. The owner of the file shall be taken from the effective owner of the process. The security and integrity levels of the file shall be taken from those of the process. The newly created file shall belong to the subtype supplied via the subtype open descriptor, stCap, providing that stCap represents a valid subtype open for writing.

3.2.3.3 Outputs

The Kernel primitive K_create shall return the following information.

ec: error conditions
result.fSeid: secure entity identifier
result.od: open object descriptor

The Kernel primitive K_create shall detect and return the following exception (error) conditions.

EX: security/integrity level of file system do not permit object at this level to be created on it
EX: too many open files
EX: the prototype SEID supplied is not for a file
EX: file system is read only
EX: file system not mounted
EX: subtype invalid, the subtype open descriptor supplied was not for a subtype open for writing
EX: unable to create file (this is a possible confinement channel)

3.2.4 Kernel Function: K device function

3.2.4.1 Inputs

The Kernel primitive K_device_function shall take the following parameters.

od: open object descriptor
f: function requested
arguments.vloc.domain: domain of parameter block containing input arguments
arguments.vloc.idSpace: space of argument parameter block
arguments.vloc.vAddr: address of argument parameter block
arguments.size: size of argument parameter block (may be omitted and have parameter block default to a standard size)
status.vloc.domain: domain of parameter block in which status will be returned
status.vloc.idSpace: space of status parameter block
status.vloc.vAddr: address of status parameter block
status.size: size of status parameter block (may be omitted and have parameter block default to a standard size)
id: asynchronous call identifier

3.2.4.2 Processing

The K_device_function primitive shall permit the process to perform special device control operations. The device shall be opened prior to the K_device_function call. The device shall be referenced only through an open object descriptor. Hence, authorized access to the device shall always be guaranteed. In addition, the requesting process must own the device. That is, the effective owner of the process must be the same as that of the device, or the process must have the privilege to change its owner. The arguments parameter block passed to the K_device_function call shall designate an area of the user process memory which contains parameters required to perform the specific function, f, requested. Thus, the particular parameters may vary depending upon which function was requested. The device specific routine shall be responsible for verifying the validity of argument parameters. In particular, certain functions may require that the process be privileged. The status parameter block shall designate an area of the process virtual memory to be used to return device specific status information. Thus, the particular parameters may vary depending upon which function was requested.

The access checking on the two parameter blocks shall be similar to that performed by the K_read_block call (below) for the arguments parameter block, and to that performed by K_write_block (below) for the status parameter block. Error returns from this call shall include invalid file descriptor and handler dependent parameter value errors in addition to the checks on the validity of the virtual addresses of the parameter blocks, the validity of the open object descriptor, and that the user owns the affected device.

Limited numbers of asynchronous K_device_function requests may be made. If the the asynchronous id field is non-null, K_device_function may return before completion of the requested operation. At the completion of the requested operation, the Kernel shall transmit a device completion pseudo interrupt, similar to that transmitted at the completion of K_read_block or K_write_block. The values returned in the area designated by the status parameter block may not be valid until the pseudo interrupt has been transmitted.

3.2.4.3 Outputs

The Kernel primitive K_device_function shall return the following information.

ec: error conditions
status: status data in status parameter block

The Kernel primitive K_device_function shall detect and return the following exception (error) conditions.

EX: invalid open object descriptor
EX: descriptor does not refer to a device
EX: user does not own device
EX: invalid parameter block (either arguments or status)
EX: invalid device operation (command)
EX: invalid device dependent parameters in area designated by arguments parameter block

3.2.5 Kernel Function: K fork

3.2.5.1 Inputs

The Kernel primitive K_fork shall take the following parameters.

None

3.2.5.2 Processing

The K_fork primitive shall create a process. The new process shall be remembered as a child of the caller (parent). The child shall be an exact duplicate of the parent. The process id of the parent shall be returned to the child. The process id of the child shall be returned to the parent. The program counter of the parent is NOT adjusted as it is in standard UNIX. The returned process id will be sufficient to distinguish parent and child. The type independent information of the child process is identical to the type independent information of the parent process.

The non-sharable segments of the process shall be copied into new segment instances for the child. The reference counts of sharable segments shall be incremented. The process local segment names shall be the same in both the parent and child, although in the case of non-sharable segments they shall refer to a different segment instance (and therefore, a different segment SEID) for the child than for the parent.

The child shall inherit the open objects of the parent. That is, each object that is open in the parent shall be opened in the child, and shall have the same local open object descriptor. The open counts of the open objects so inherited shall be incremented to reflect the fact that another process (the child) has them open. If the parent has an object open for exclusive use, the K_fork call shall fail, preventing two processes from having simultaneous access to the same exclusive use object.

An error shall be returned if the number of available processes has been exhausted. Analysis will be required to determine the bandwidth of this binary information channel and to select an appropriate solution.

3.2.5.3 Outputs

The Kernel primitive K_fork shall return the following information.

childSeid: process SEID (the parent gets childSeid, and the child gets the process SEID of the parent)
ownSeid: process SEID (the parent gets parentSeid, and the child gets childSeid)
ec: error conditions

The Kernel primitive K_fork shall detect and return the following exception (error) conditions.

EX: exclusive use files open
EX: unable to create new process (possible information channel)

3.2.6 Kernel Function: K get file status

3.2.6.1 Inputs

The Kernel primitive K_get_file_status shall take the following parameters.

fSeid: file seid

3.2.6.2 Processing

K_get_file_status shall return the type dependent information associated with a file. The exact format and information content of the file status block shall be left to the discretion of the development contractor subject to

Government review and approval. The invoking process must have read access to the file with respect to the manditory security and integrity rules only. No discretionary access checking shall be performed. (This is required for faithful emulation of the UNIX construct of lock files which are created with no discretionary access permissions.)

3.2.6.3 Outputs

The Kernel primitive K_get_file_status shall return the following information.

ec: error conditions
result.nBlocks: the size of the file in blocks
result.linkCount: the number of references to the file in higher level constructs (e.g. directories or equivalent)
result.timeLastMod: the time of last modification to the file
result.subtype: the subtype to which the file belongs (a SEID)
result.openForWriting: a Boolean flag indicating that this file is open for writing (TRUE) or was open for writing when the system was halted. Such information is needed for higher order file recovery after unplanned system halts. A value of FALSE shall mean that the file had been normally closed at the time of system shutdown.

The Kernel primitive K_get_file_status shall detect and return the following exception (error) conditions.

EX: object does not exist or is not a file, device or subtype
EX: attempt to read a higher security level object (the actual return value is as if the object did not exist)
EX: attempt to read a lower integrity level object (the actual return value is as if the object did not exist)

3.2.7 Kernel Function: K get object level

3.2.7.1 Inputs

The Kernel primitive K_get_object_level shall take the following parameters.

seid: the seid of the desired object

3.2.7.2 Processing

The primitive K_get_object_level shall return the type independent information for an object.

3.2.7.3 Outputs

The Kernel primitive K_get_object_level shall return the following information.

ec: error conditions

otii.nd.securityCategory: security classification category of object (e.g. TOP SECRET, UNCLASSIFIED)
otii.nd.securityComp: security compartments of the object
otii.nd.integrityCategory: integrity classification category of the object
otii.nd.integrityComp: integrity compartments of the object (now, always null)
otii.da: discretionary access permissions for the object (read, write, execute/search for the owner, group and others)
otii.owner: object owner (a user ID)
otii.group: object group (a group ID)
otii.priv: the privileges associated with the object

The Kernel primitive K_get_object_level shall detect and return the following exception (error) conditions.

EX: object does not exist
EX: attempt to read a higher security level object (the actual return value is as if the object did not exist)
EX: attempt to read a lower integrity level object (the actual return value is as if the object did not exist)

3.2.8 Kernel Function: K get process status

3.2.8.1 Inputs

The Kernel primitive K_get_process_status shall take the following parameters.

objSeid: SEID of process about which to get status

3.2.8.2 Processing

The K_get_process_status call shall return the type dependent information about a process. A process may successfully request information of processes that it can access given its level and the level of the target process.

3.2.8.3 Outputs

The Kernel primitive K_get_process_status shall return the following information.

ec: error conditions
ps.self: the SEID of the process (should be the same as objSeid)
ps.parent: the SEID of the parent process (The parent of a process, p, is the process which issued the Kernel call that brought process p into existence.)
ps.family: the process family identifier
ps.realUser: the real user ID
ps.realGroup: the real group ID
ps.pc: program counter (PDP-11 specific)
ps.ps: processor status word (PDP-11 specific)

ps.pil: pseudo-interrupt level
ps.advPrio: advisory priority
ps.timerAlarm: the time remaining before a timer interrupt will occur
ps.supervisorTiming: execution time in the supervisor domain (PDP-11 specific) (This need not be an exact timing.)
ps.userTiming: execution time in the user domain (PDP-11 specific)
ps.timTog: Boolean indicating whether reptitive timer interrupts are to be generated

The Kernel primitive K_get_process_status shall detect and return the following exception (err r) conditions.

EX: process does not exist or is inaccessible

3.2.9 Kernel Function: K get segment status

3.2.9.1 Inputs

The Kernel primitive K_get_segment_status shall take the following parameters.

segSeid: seid of the desired segment

3.2.9.2 Processing

K_get_segment_status shall return the type dependent information associated with a segment. A process may successfully obtain type dependent status about any segment from which information could flow to the process. No discretionary access checking shall be performed.

3.2.9.3 Outputs

The Kernel primitive K_get_segment_status shall return the following information.

ec: error conditions
ss.gl.sharable: Boolean indicating whether or not the segment may be shared
ss.gl.lock: Boolean incicating whether or not the segment is to be locked in memory
ss.gl.sticky: Boolean indicating whether or not the segment is to be retained in the swap area after there are no more active references to it
ss.gl.memAdvise: Boolean indicating whether or not it is expected that I/O will be done to this segment (this is merely advisory, I/O can be done to any segment which is accessible in the desired mode)
ss.gl.executable: Boolean indicating whether or not the segment contains executable data
ss.gl.direction: growth direction for the segment (Up or down) (PDP-11 specific)
ss.size: the size of the segment, in the PDP-11 this is in bytes

The Kernel primitive K_get_segment_status shall detect and return the following exception (error) conditions.

EX: segment doesn't exist or is inaccessible

3.2.10 Kernel Function: K interrupt return

3.2.10.1 Inputs

The Kernel primitive K_interrupt_return shall take the following parameters.

none

3.2.10.2 Processing

K_interrupt_return shall provide an atomic return operation from pseudo interrupts. It can be thought of as being analogous to the PDP-11 RTI and RTT instructions. When a pseudo-interrupt occurs, the program counter, processor status word. and current pseudo-interrupt level shall be saved in a pseudo-interrupt vector for the particular type of pseudo-interrupt which occurred. In the PDP-11 implementation, these vectors shall be located at fixed locations in the supervisor domain. The K_interrupt_return call shall restore the process state from these saved values. Because the interrupted process state is accessible to the process, the K_interrupt_return call shall check the saved state prior to restoring it. The process shall not be permitted to increase its privileges or accessible domains. (Similar checking takes place in the processing of the pseudo interrupt itself.) For example, in the PDP-11 implementation, the process shall not be allowed to set the either mode (current or previous) to kernel or to alter the processor priority and general register set information.

3.2.10.3 Outputs

The Kernel primitive K_interrupt_return shall return the following information.

none

3.2.11 Kernel Function: K invoke

3.2.11.1 Inputs

The Kernel primitive K_invoke shall take the following parameters.

immSeid: segment seid of intermediary process segment
arg: segment designator specifying argument segment

3.2.11.2 Processing

The purpose of the K_invoke primitive is the invocation of potentially privileged software. The effect of this call shall be to replace the existing segment map (including the executing text segment) by a new process image. The segments currently active may or may not be released, at the option of the development contractor, subject to Government approval. The new process image shall have only the intermediary segment and the argument segment active (mapped in). Arguments for use by the intermediary process may be placed in the argument segment. The exact format of the argument segment is determined by the particular intermediary specified in the call. The argument segment may be used by the intermediary as a scratch pad as the intermediary builds any other segments it requires. It shall be the responsibility of the newly executing program (the intermediary) to create its own working segments. The privileges of the process shall be set to those associated with the intermediary segment. In the PDP-11/70 implementation, the intermediary shall be mapped into location 0 of the supervisor domain instruction space, and the argument segment shall be mapped into location 0 of the supervisor mode data space. Thus, the intermediary must be coded as a separated instruction/data space program. The intermediary may perform any arbitrary function. Thus, applications of the KSOS Kernel may elect to create specialized intermediaries to perform specific functions. The only pre-defined intermediary is the Process Bootstrapper, described next.

3.2.11.3 Process Bootstrapper Processing

The Process Bootstrapper segments shall implement a trusted process whose sole purpose is the creation of other, potentially trusted, environments by replacing itself with image from the prototype file whose name is passed in as an argument. The Process Bootstrapper shall have the following privileges:

a. to set privileges

b. to set the effective owner

c. to set the sticky bit

d. to set the lock (no swapping) bit

e. to set the security and integrity level

f. to realize execute permissions (i.e. use the execute permissions for read access attempts)

Using the parameters specified in the argument segment, the Process Bootstrapper shall build a new set of process segments conforming to the process prototype file. The privileges for the new environment shall be obtained from the process prototype file. If the prototype file has no privileges associated with it, the new environment shall be unprivileged. If the prototype file specifies that it is to execute in a different discretionary access domain, the bootstrap shall change the effective user and/or group of the process to the owner of the prototype file. The new trusted process shall then be set into execution by the Process Bootstrapper. Note that a completely trusted path exists from the K_invoke call, through process construction, to the execution of the trusted software.

The use of the K_invoke call shall not be limited to the invocation of trusted processes. Untrusted processes may also be executed through the K_invoke call. If change of discretionary access domain or privilege is not required by the type dependent information associated with the process prototype file, the process bootstrap shall simply remove all privileges prior to setting the new image into execution.

The privilege information associated with a process prototype file shall be controlled by the Privilege Control Process, a restricted use program discussed in the Non-Kernel Security-Related Software CPCI Specification [NKSR 78].

3.2.11.4 Outputs

The Kernel primitive K_invoke shall return the following information.

ec: error conditions

The Kernel primitive K_invoke shall detect and return the following exception (error) conditions.

EX: invalid argument segment designator
EX: argument segment is sharable
EX: argument segment is not writable
EX: intermediary segment does not exist or is inaccessable
EX: intermediary segment does not have execute permissions for this user
EX: argument segment could not fit into memory available to it
EX: intermediary segment could not fit into memory available to it

The intermediary may detect additional error conditions associated with the particular arguments supplied. Due to the fact that the process environment is discarded prior to the activation of the intermediary, these errors would normally be reported to the parent of the process via an IPC message from the intermediary. The process bootstrap shall report the following exception (error) conditions:

EX: prototype file is inaccessible
EX: format of argument segment is invalid
EX: arguments to invoked process too large

3.2.12 Kernel Function: K_link

3.2.12.1 Inputs

The Kernel primitive K_link shall take the following parameters.

fSeid: SEID of file affected

3.2.12.2 Processing

K_link shall increment the reference count of a Kernel file. The reference count may be used to indicate the number of UNIX directory entries which point to this SEID. Applications of the KSOS Kernel which do not use the UNIX directory structure and semantics may use the reference count for other purposes. The reference count may only be incremented by processes privileged to issue this call. Such processes should be carefully designed to reduce the bandwidth of the resultant confinement channels and to preserve the integrity of the higher level directory structure, if any. The security and integrity checking on K_link shall be as if the user were reading and writing the file. No discretionary access checking shall be performed. Thus, the processes privileged to use K_link may implement whatever discretionary checking they chose to.

For example, the Directory Manager of the UNIX Emulator [Emulator 78], when emulating the UNIX link(II) call, checks that the directory in which the new entry is being place is writable by this user prior to making the K_link call.

3.2.12.3 Outputs

The Kernel primitive K_link shall return the following information.

ec: error conditions

The Kernel primitive K_link shall detect and return the following exception (error) conditions.

EX: insufficient privilege
EX: object does not exist or is inaccessible from security or integrity considerations
EX: object is not a file
EX: object is in a file system which has been mounted as read only

3.2.13 Kernel Function: K mount

3.2.13.1 Inputs

The Kernel primitive K_mount shall take the following parameters.

dev: device SEID
leaf: file system new SEID
root: file system old SEID
readOnly: Boolean read only flag TRUE => read only, no writing is permitted on the name space partition being mounted, FALSE => writing allowed

3.2.13.2 Processing

The K_mount call performs two functions:

a. associating a particular name space partition with a device

b. causing all references to the "leaf SEID" to refer to a SEID ("root SEID") on the newly mounted name space partition

In KSOS, physical disk volumes may be logically subdivided into "extents". Each extent containing a KSOS file system shall belong to a different name space partition. Thus, the SEID of a file is unique across all the file systems that could be mounted. It shall be the responsibility of the System Administrator to assure that the name space partition assignment are unique. The Immigration Officer software [NKSR 78] maintains a data base of name space partitions presently assigned. When a extent is mounted, the Kernel shall update an internal data base that tells it on which device the SEID's in the mounted name space partition may be found. The second facet of the mount mechanism is the logical connection of the newly mounted file system into the rest of the KSOS file system. This shall be accomplished by mapping all references to "leaf SEID", a SEID of an existing, currently available (i.e. the old SEID's file system must itself be presently mounted and the leaf SEID must exist and be inactive) file to a SEID on the newly mounted file system, the "root SEID". It shall be possible to mount a file system as read only, in which case no writing may be done to any file in that file system.

Each extent contains type independent information. The security and integrity levels of the extent shall be interpreted as the maximum levels allowed for any file on the extent. The discretionary access information shall determine who may mount that particular extent. To mount a extent the user must have write permission to it. This shall prevent users from mounting the private extents of other users or inactive system extents. The security level and discretionary access permissions on the devices shall determine who may mount any extent on that device. The disk devices shall not be reserved only for KSOS file systems. A given disk device may be assigned for private, non-file system use.

The following checks shall be performed prior to performing the mount:

a. the process must be privileged to issue the call

b. the user must own the device

c. the user must have write permission to the extent

d. the security level of the device must be equal to the security level of the extent

e. both the root SEID and the leaf SEID must be files (although, normally they would both be of the subtype directory)

f. the leaf SEID must be inactive (not opened)

g. the device SEID must be a block organized device (disk)

h. the file system must not be presently mounted

i. sufficient Kernel mount table space must exist (this is a low bandwidth confinement channel)

The Kernel primitive K_mount would normally be called by privileged Non-Kernel Security-Related Software which would make additional checks before issuing the K_mount call.

3.2.13.3 Outputs

The Kernel primitive K_mount shall return the following information.

ec: error conditions

The Kernel primitive K_mount shall detect and return the following exception (error) conditions.

EX: insufficient privilege level
EX: the leaf file does not exist or is inaccessible for security or integrity reasons
EX: the extent, dev, does not exist or is inaccessible for security or integrity reasons
EX: the leaf file cannot be written by this user (discretionary access)
EX: the dev extent cannot be written by this user
EX: the leaf is not a file
EX: the root is not a file
EX: dev is not an extent
EX: the extent is already mounted
EX: this user is not the owner of the extent
EX: the extent is open
EX: the leaf is open
EX: no more room in the mount table

3.2.14 Kernel Function: K nap

3.2.14.1 Inputs

The Kernel primitive K_nap shall take the following parameters.

timeOut: time which should pass before process should be rescheduled

3.2.14.2 Processing

K_nap shall be a mechanism for explicitly giving up the processor when a higher level blocking condition occurs. This situation occurs when, for example, processes implementing semaphores on top of the Kernel become logically blocked on a semaphore. K_nap provides an alternative to busy waiting for the semaphore. The timeOut argument shall be the incremental time before which the process should not be rescheduled by the Kernel. Processes using K_nap should check that the logical condition for which they were waiting has occurred when they are activated.

3.2.14.3 Outputs

The Kernel primitive K_nap shall return the following information.

none

The Kernel primitive K_nap shall detect and return the following exception (error) conditions.

none

3.2.15 Kernel Function: K_open

3.2.15.1 Inputs

The Kernel primitive K_open shall take the following parameters.

oSeid: SEID of the object to be opened
om: requested access mode (read, write, read and write, write with exclusive use, or read and write with exclusive use)
stCap: explicit subtype capability (optional, open object descriptor)

3.2.15.2 Processing

The Kernel open primitive shall be the Kernel analog of the UNIX user interface open call. SEID interpretation shall be performed by the Kernel. A successful K_open call may be thought of as being equivalent to successfully granting the requested access capability to the object. The security and integrity levels of the calling process must permit the flow of information to or from the object to be opened, depending upon the open mode. For example, to open for reading, the security level of the object must be at or below the level of the process. To open for writing, the security level of the object must be at or above the level of the process. The caller shall have the appropriate explicit or implicit discretionary permission to the subtype of the file. Explicit subtype capability shall be passed to the Kernel in the form of an open object descriptor returned from a previous, successful K_open on the file subtype SEID. Implicit subtype capability shall be derived by checking the requested access mode against the valid access modes permitted to the effective group and user ID of the process. That is, if the subtype discretionary access permissions allow accesses in the requested mode, no explicit subtype capability need be supplied. For example, if the subtype discretionary access permissions allow read access by all users and the user is opening the object for reading, the subtype capability need not be supplied. The caller shall have the appropriate discretionary access permission to the file as granted by the owner of the file. An open object descriptor (virtualized on a per process basis) shall be returned for subsequent references to the object.

The valid combinations of access modes shall be:

a. read

b. write

c. read and write

d. write with exclusive use

e. read and write with exclusive use

As indicated above, if exclusive use is requested, write access must also be requested. This requirement is motivated by security considerations.

Requests for open with exclusive use shall be blocking. (Failure returns from unsuccessful exclusive use opens create clandestine information channels.) They shall remain blocked until the outstanding open on the object has been closed or until a (system default) timeout expires. Non-exclusive use opens shall also block when attempting to open a file with outstanding exclusive use access. The number of files opened for exclusive use shall be limited to one per process to prevent a class of deadlocks. Exclusive use opens shall not be passed to the child process as a result of a fork operation. The fork shall fail until the exclusive use file has been closed.

3.2.15.3 Outputs

The Kernel primitive K_open shall return the following information.

ec: error condition
od: open descriptor (process local)

The Kernel primitive K_open shall detect and return the following exception (error) conditions.

EX: object does not exist, or is inaccessible from security or integrity considerations
EX: object link count is zero
EX: subtype capability that was supplied was not for a subtype
EX: subtype capability not supplied, but object was in a subtype
EX: subtype capability supplied was not opened in the modes that are being requested for the object, or was not the same subtype as the object
EX: no discretionary access permission
EX: device on which object resides has been mounted as read only, and write access requested
EX: exclusive use requested, but no write access requested
EX: another process has the object open for exclusive use
EX: exclusive use requested and another file already opened for exclusive use by this process
EX: too many open objects for this process

3.2.16 Kernel Function: K_post

3.2.16.1 Inputs

The Kernel primitive K_post shall take the following parameters.

receiver: SEID of process to which message is to be sent
pseudoInt: Boolean flag TRUE => cause pseudo interrupt in receiving process FALSE => no pseudo interrupt
msg: IPC message text (the exact size of the text, and whether this text is of fixed or varible size is left to the discretion of the development

contractor, subject to Government approval)

3.2.16.2 Processing

K_post shall send a short message to another process. A pseudo interrupt shall be asserted at the destination process, if selected, and if the receiving process has IPC pseudo interrupts enabled (i.e. that its pseudo-interrupt level is sufficiently low to allow pseudo interrupts). A header shall be attached to the message indicating the SEID of the originating process.

The type independent information for the two processes shall be used to determine rights of the originating process to communicate with the the destination.

3.2.16.3 Outputs

The Kernel primitive K_post shall return the following information.

ec: error conditions

The Kernel primitive K_post shall detect and return the following exception (error) conditions.

EX: destination process does not exist or is inaccessible
EX: destination process cannot accept any more messages

Because the last exception condition can be used as an information channel, the bandwidth of this channel shall be limited by forcing a wait before return.

3.2.17 Kernel Function: K read block

3.2.17.1 Inputs

The Kernel primitive K_read_block shall take the following parameters.

od: open object descriptor
blockNo: the logical block number in the file at which to start reading
duFile.vloc.domain: domain of block into which the file will be read
duFile.vloc.idSpace: space of block
duFile.vloc.vAddr: starting address of the block
duFile.size: length (in bytes) of the block, i.e., the maximum number of bytes to read
as: asynchronous call identifier

3.2.17.2 Processing

K_read_block shall retrieve block(s) of data from a previously opened file. An error shall be reported if the file was not opened for reading. The security conditions and discretionary access rights shall be checked at the time of K_open. Error and end of file conditions shall be encoded into the return value from K_read_block. Other errors shall include bad parameter values and physical

I/O errors.

Only block mode I/O shall be supported at the Kernel level. Stream I/O can be implemented in the UNIX Emulator by caching the data blocks in Emulator level buffers. The Emulator would also maintain the seek or file pointers. The block length parameter shall specify the maximum number of bytes to be read from the file or device. This parameter value may be device dependent and validity shall be checked by the specific device driver. Certain devices, such as terminals, can accept any block length. For such devices, the "block" consists of bytes from the input stream up until the occurrence of one of a user-controllable (via K_device_function) set of break characters or until the vector is full. The Kernel shall check for the validity of the vector in which the data from the file is to be placed. These checks include checking that the vector is valid and accessible (i.e. can be written) and that the entire vector lies within one currently mapped in segment.

The Kernel file system shall allow "sparse" files, i.e. files in which not all blocks have been written into. When one of these unwritten blocks is read zeros shall be placed in that part of the returned data vector.

Limited numbers of asynchronous K_read_block requests may be made. If the the asynchrony id field is non-null, K_read_block may return before completion of the requested operation. At the completion of the requested operation, the Kernel shall transmit an IPC message to the requesting process containing (at least): the SEID of device, the asynchrony id, the number of bytes read, and other device dependent status information.

3.2.17.3 Outputs

The Kernel primitive K_read_block shall return the following information.

ec: error conditions
ef: end of file. One of the blocks requested was larger than the number of blocks in the file. This condition may be treated as an error condition. All blocks that exist will be returned.
da: vector of data in the block (not an explicit return parameter in the formal specifications in Appendix A)
result.bytesRead: actual number of bytes read
result.errst.devIndep: device independent I/O completion status
result.erst.devDep: device dependent I/O completion status

The Kernel primitive K_read_block shall detect and return the following exception (error) conditions.

EX: invalid open object descriptor
EX: object is a subtype
EX: object not opened for reading
EX: object is a terminal, and this path is not the current path
EX: size of the request was bad (i.e. below minimum, above maximum, or not an integral multiple of device dependent block size)
EX: bad block number
EX: end of file
EX: bad parameter block specification (entire parameter block must fit

within one mapped in segment which can be written into)

3.2.18 Kernel Function: K receive

3.2.18.1 Inputs

The Kernel primitive K_receive shall take the following parameters.

timeOut: if no messages have been received within timeOut, return

3.2.18.2 Processing

K_receive shall suspend the execution of a process until the receipt of an IPC message or until a time out. The return value shall indicate the condition which caused the process to be rescheduled.

The first message in the queue of received IPC messages shall be returned. If the timeOut expires before any pseudo-interrupts occur, the error code shall so indicate.

3.2.18.3 Outputs

The Kernel primitive K_receive shall return the following information.

ec: error conditions
msg.sender: process SEID of sender (filled in by the Kernel)
msg.text: data from sender

The Kernel primitive K_receive shall detect and return the following exception (error) conditions.

EX: timeOut expired with no message received

3.2.19 Kernel Function: K release process

3.2.19.1 Inputs

The Kernel primitive K_release_process shall take the following parameters.

rSeid: SEID of process to release (defaults to self)

3.2.19.2 Processing

K_release_process shall deallocate all of the Kernel level resources associated with the named process. Only a process with the same owner or a process privileged to change its owner may issue a K_release_process for another process. The effects of K_close for all open files and of a K_release_segment for all the segments of the process shall occur. Shared segments shall remain intact unless the reference count to the segment has reached zero. Segments with a zero reference count shall be deallocated unless they have been created to be "sitcky". Files shall be deallocated if their reference counts and open counts are zero. The process id shall become unknown.

3.2.19.3 Outputs

The Kernel primitive K_release_process shall return the following information.

ec: error conditions

The Kernel primitive K_release_process shall detect and return the following exception (error) conditions.

EX: not the owner of rSeid and no privilege to change ownership

3.2.20 Kernel Function: K release segment

3.2.20.1 Inputs

The Kernel primitive K_release_segment shall take the following parameters.

segDes: segment name (process local designator)

3.2.20.2 Processing

The primitive K_release_segment shall release the Kernel level resources *associated with the specified segment.* The segment shall not be deleted if other processes are still using it or if its sticky bit is set. An error condition shall be returned if the segment name is unknown within the process.

3.2.20.3 Outputs

The Kernel primitive K_release_segment shall return the following information.

ec: error conditions

The Kernel primitive K_release_segment shall detect and return the following exception (error) conditions.

EX: invalid segment designator

3.2.21 Kernel Function: K remap

3.2.21.1 Inputs

The Kernel primitive K_remap shall take the following parameters.

in: designator name of segment to be mapped in (may be null)

vl.domain: domain new segment to be mapped into
vl.idSpace: Instruction space of Data space segment placement
vl.vAddr: virtual address within domain of segment
da: requested access mode (read, write, read and write)
vlFlg: Boolean flag indicating whether (TRUE) or not (FALSE) virtual location of this segment should be changed in accordance with the virtual location specified in the call
daFlg: Boolean flag indicating whether or not discretionary access modes should be changed
out: designator name of segment to be mapped out (may be null)
outSize: new size of outgoing segment
osFlg: Boolean indicating whether or not the size of the segment should be changed

3.2.21.2 Processing

The K_remap primitive shall permit the process to change its segment map. The outgoing segment shall no longer be directly addressable by the process through machine instructions. The incoming segment shall become directly addressable by the process. The outgoing segment shall not be released (deleted.) However, the memory management hardware of the segment to be removed from the current mapped set may be used to satisfy the hardware requirements of the incoming segment. When a process segment is mapped into the current addressable set of segments, it shall occupy the virtual address vector defined by the arguments to K_remap. Either but not both of the segment designators may be null. If both are null the call shall have no effects. The incoming segment must fit into the virtual memory and memory management resources available after the outbound segment is unmapped. If it does not, or if any of the other error conditions occur, the call shall have no effect on the segment mapping.

If the alter virtual location flag (vlFlg) is TRUE, the incoming segment shall be mapped into the location specified as arguments to the call, and its status information adjusted to reflect this as a permanent change. Otherwise, the segment shall be mapped into the location specified in its permanent status information.

If the alter discretionary access information flag (daFlg) is TRUE, the modes in which this process will access the segment shall be checked against the permitted access modes for the segment, and if allowed, will become the access modes for the segment. This may alter the settings of memory management hardware when the segment is mapped back in.

If the alter size flag (osFlg) is TRUE, the size of the outbound segment shall be set to the value outsize. The expansion or truncation of the segment is performed at the end of segment specified by the growth attribute of the segment specified when built. Expanded parts of segments shall be filled with zeros. The size change can only be applied to segments that are not sharable.

3.2.21.3 Outputs

The Kernel primitive K_remap shall return the following information.

ec: error conditions

The Kernel primitive K_remap shall detect and return the following exception (error) conditions.

EX: segment "in" is not one of the active segments for this process
EX: segment "out" is not mapped in
EX: segment "in" is already mapped in
EX: new segment discretionary access modes are for write only
EX: atempt to change ths memory space of a segment
EX: virtual address space overlap or insufficient memory mapping hardware to satisfy request
EX: size change requested on sharable segment
EX: new discretionary access modes not allowed to this segment

3.2.22 Kernel Function: K rendezvous segment

3.2.22.1 Inputs

The Kernel primitive K_rendezvous_segment shall take the following parameters.

segSeid: name (SEID) of segment with which to rendezvous
vl.domain: domain for the segment, i.e. which domain the segment is to be mapped into
vl.idSpace: space for the segment
vl.vAddr: virtual address for the segment
da: access type requested (read | write)

3.2.22.2 Processing

The Kernel call K_rendezvous_segment shall be the mechanism by which processes are able to share segments. If the segment requested exists and is accessible, it shall be mapped into the processes address space as requested, providing that the requested mapping information is valid. The Kernel shall check that the segment may be mapped into the process issuing the K_rendezvous_segment call. The checks shall include:

a. that the segment seid is active

b. that the segment may be shared

c. that the security/integrity level of the process allows it to access the segment

d. that the discretionary access for the segment allows it to be accessed in the requested way

e. that the virtual address supplied is valid

3.2.22.3 Outputs

The Kernel primitive K_rendezvous_segment shall return the following information.

ec: error conditions
segd: process local name for the segment

The Kernel primitive K_rendezvous_segment shall detect and return the following exception (error) conditions.

EX: segment not active or cannot be accessed from the process security and integrity level
EX: requested access mode was write only
EX: discretionary access denied
EX: segment already known to this process (may or may not be mapped in)
EX: virtual memory address space would be exceeded or overlapped
EX: too many segments for this process

3.2.23 Kernel Function: K secure terminal lock

3.2.23.1 Inputs

The Kernel primitive K_secure_terminal_lock shall take the following parameters.

devSeid: SEID of terminal ("desired" path)

3.2.23.2 Processing

The K_secure_terminal_lock call shall provide a mechanism to control which of several paths to a terminal is currently active. The call may be successfully invoked only by processes privileged to do so. For each terminal, there are several logical devices, one of which is normally reserved for use by trusted software (the "secure device"). The input/output stream is switched between these logical devices by the Secure Server (part of the Non-Kernel Security Related Software). The user may send the secure attention character to the computer. This causes the Kernel to send an IPC message to the distinguished process, the Secure Initiator [NKSR 1978], and to "freeze" the terminal by terminating all in-progress operations on the terminal and refusing to accept new requests on any path to the terminal. Freezing means that all paths are blocked; no characters may be read or written to (from) the terminal by the active, untrusted processes. The secure attention character shall always be active, and shall be site selectable as a system generation parameter. The requirement that the secure attention character always be active means that the setting of terminal speed and parity must be performed by trusted software.

In the frozen state, no path is active, but the Secure Initiator has been informed of the situation. No further action can take place on the terminal until the trusted software causes a K_secure_terminal_lock to take place, thus "thawing" a path for the use of the Secure Server.

Now, only the secure path is active, thus the user can be assured that he (she) is communicating only with trusted software, and not some untrusted, masquerading process. Only the trusted software may thaw a user path (call K_secure_terminal_lock with a SEID of a user path to the terminal) which then deactivates the secure path, and unblocks a normal path.

Each path to a terminal shall have a separate SEID and separate set of Type Independent Information. The SEIDs of a set of paths to a given terminal are dense; the first SEID of a set is designated the secure path. The Secure Server may change the level of a path by the use of the K_set_object_level function, but only if the path is currently not open by any process. Thus the tranquility property is enforced for all paths, and it is thus not possible for the level of a path to change while it is associated with a process.

A unique exception status shall be returned when a process attempts I/O on a frozen path. It is the responsibility of the process to wait a short period and try again if it wishes to wait for the path to be unfrozen (this occurs whn the user changes level at his terminal and wants to leave a running process behind).

Because the processes are not killed when they are unable to write to the terminal at its new level, it is possible to return to level later and resume as if nothing had happened.

3.2.23.3 Outputs

The Kernel primitive K_secure_terminal_lock shall return the following information.

ec: error conditions

The Kernel primitive K_secure_terminal_lock shall detect and return the following exception (error) conditions.

EX: SEID does not refer to a terminal
EX: process is not privileged to use secure terminal interface
EX: SEID does not exist

3.2.24 Kernel Function: K set file status

3.2.24.1 Inputs

The Kernel primitive K_set_file_status shall take the following parameters.

fseid: seid of desired object
newStatus.nBlocks: block size of the file (should not be modified without corresponding alteration of any on-disk fill block allocation information)
newStatus.linkCount: number of references to the file (used by UNIX Emulator and Directory Manager to indicate number of directory entries for the file)
newStatus.timeLastMod: time of the last modification to the file

newStatus.subtype: the subtype (if any) to which the file belongs
newStatus.openAtCrash: Boolean flag indicating whether (TRUE) or not (FALSE) the file is open for writing or was open when the system halted (used for recovery of higher level constructs like directories)

3.2.24.2 Processing

K_set_file_status shall set the file object type dependent information. This call should be used primarily for maintenance functions. The intent is to allow programs performing recovery and maintenance functions for higher level constructs, e.g. UNIX directories to have the ability to manipulate file status information. Manintenance and recovery of the basic Kernel file system cannot be performed completely through this call. Rather, these basic file system recovery actions may require manipulation of the disk data directly. Naturally, such programs must be designed and implemented with the same thoroughness and techniques used for the Kernel implementation, and their use must be restricted to authorized maintenance personnel. The process shall require the privilege to set file status to successfully use this call.

3.2.24.3 Outputs

The Kernel primitive K_set_file_status shall return the following information.

ec: error conditions

The Kernel primitive K_set_file_status shall detect and return the following exception (error) conditions.

EX: either file does not exist or is inaccessible from security or integrity considerations
EX: no discretionary access permissions to file
EX: not the file owner
EX: not privileged to change file control information

3.2.25 Kernel Function: K set object level

3.2.25.1 Inputs

The Kernel primitive K_set_object_level shall take the following parameters.

objseid: seid of desired object
level.nd.securityCat: new security classification category of object
level.nd.securityComp: new security compartment set
level.nd.integrityCat: new integrity classification category of object
level.nd.integrityComp: new integrity compartment set (now always null, may be omitted)
level.da: new discretionary access permissions for object (read, write, execute for owner, group and others)
level.owner: new owner for the object

level.group: new group of the object
level.priv: new privileges for the object

3.2.25.2 Processing

The K_set_object_level primitive shall set the type independent information for an object.

The owner of the file shall be capable of changing the discretionary access information. Processes with the privilege to set object level (see 3.1.1.2.5) shall be capable of changing:

a. the user which owns the object

b. the group which owns the object

c. the discretionary permissions

d. the security level (security category and compartments)

e. the integrity level (integrity category and (presently null) compartments.)

Processes shall require an additional privilege to set the privileges of an object. If the appropriate conditions are not met to change the object level/access information, only the information which the user is privileged to change shall be altered.

3.2.25.3 Outputs

The Kernel primitive K_set_object_level shall return the following information.

ec: error conditions

The Kernel primitive K_set_object_level shall detect and return the following exception (error) conditions.

EX: object does not exist or is inaccessible due to security or integrity considerations
EX: insufficient privilege to change mandatory access control information
EX: only owner or privileged process can change file owner or discretionary permissions
EX: improper security level, integrity level, access mode, owner or domain

3.2.26 Kernel Function: K set process status

3.2.26.1 Inputs

The Kernel primitive K_set_process_status shall take the following parameters.

procSeid: SEID of process whose status is to be set
ps.self: the SEID of the process (cannot be altered, included for symmetry with K_get_process_status)
ps.parent: the SEID of the parent process (cannot be altered, included for symmetry)
ps.family: the process family identifier
ps.realUser: the real user ID
ps.realGroup: the real group ID
ps.pc: program counter (PDP-11 specific)
ps.ps: processor status word (PDP-11 specific)
ps.pil: pseudo-interrupt level
ps.advPrio: advisory priority
ps.timerAlarm: the time remaining before a timer interrupt will occur
ps.supervisorTiming: execution time in the supervisor domain (PDP-11 specific) (This need not be an exact timing.) (cannot be altered, included for symmetry)
ps.userTiming: execution time in the user domain (PDP-11 specific) (cannot be altered, included for symmetry)
ps.timTog: Boolean indicating whether reptitive timer interrupts are to be generated

3.2.26.2 Processing

The K_set_process_status call shall permit the process to change those type dependent parameters that are not controlled by other primitives.

The K_set_process_status Kernel call shall supply an advisory scheduling priority to the Kernel level scheduler. The Kernel may elect to adjust the advisory priority to guarantee equitable service to all processes. Only processes privileged to set privileges may alter their privileges.

The notion of real and effective user identification shall be retained at the Kernel level because these identifiers determine the access permissions extended to a process. The effective user and group ID's are part of the type independent information for the process, because they are what determine the discretionary access rights. The real user and group ID's are part of this type dependent information.

The timer toggle and pseudo interrupt level control the pseudo interrupt mechanism. If the timer toggle is TRUE, a pseudo interrupt shall be generated every clock tick (machine dependent time unit). This mechanism may be used for periodic sampling of user mode program counter values for the construction of execution profiles. The pseudo interrupt level is analogous to the hardware interrupt level. Pseudo-interrupts shall be transmitted to the process only if the level of the pseudo interrupt is above the level of the process. The process shall not be able to raise the pseudo interrupt level sufficiently to prevent hardware error pseudo interrupts.

3.2.26.3 Outputs

The Kernel primitive K_set_process_status shall return the following information.

ec: error conditions

The Kernel primitive K_set_process_status shall detect and return the following exception (error) conditions.

EX: the process does not exist, or is inaccessible due to security, integrity or discretionary access considerations

3.2.27 Kernel Function: K set segment status

3.2.27.1 Inputs

The Kernel primitive K_set_segment_status shall take the following parameters.

segSeid: seid of the desired segment
st.gl.sharable: Boolean indicating whether or not the segment may be shared
st.gl.lock: Boolean incicating whether or not the segment is to be locked in memory (requires privilege to set)
st.gl.sticky: Boolean indicating whether or not the segment is to be retained in the swap area after there are no more active references to it (requires privilege to set)
st.gl.memAdvise: Boolean indicating whether or not it is expected that I/O will be done to this segment (this is merely advisory, I/O can be done to any segment which is accessible in the desired mode)
st.gl.executable: Boolean indicating whether or not the segment contains executable data
st.gl.direction: growth direction for the segment (Up or down) (PDP-11 specific)
st.size: the size of the segment, in the PDP-11 this is in bytes

3.2.27.2 Processing

The Kernel primitive K_set_segment_status supports modification of the type dependent information of a segment. The invoking process shall have appropriate privilege in order to modify the "sticky" flag or the "lock" flag.

3.2.27.3 Outputs

The Kernel primitive K_set_segment_status shall return the following information.

ec: error conditions

The Kernel primitive K_set_segment_status shall detect and return the following exception (error) conditions.

EX: segment doew not exist or is inaccessible from security or integrity considerations
EX: discretionary access violation (user must have read and write access)
EX: segment is sharable

EX: growth direction is different than curent value
EX: attempt to alter "sticky" without privilege to do so
EX: attempt to alter "lock" without privilege to do so

3.2.28 Kernel Function: K signal

3.2.28.1 Inputs

The Kernel primitive K_signal shall take the following parameters.

procSeid: SEID of process to be signaled
sigVal: value of the signal

3.2.28.2 Processing

The K_signal primitive shall provide a means for privileged processes to transmit a high priority pseudo-interrupt to a process. K_signal differs from the K_post IPC mechanism in several ways. First, K_signal always generates a pseudo interrupt. The pseudo-interrupt level of the K_signal pseudo-interrupt is above that of normal IPC. Second, the K_signal pseudo interrupt will abort long running Kernel calls (i.e. terminal I/O) which receiving the K_post mechanism does not. Third, a much smaller message is transmitted. The intended use of K_signal is to provide a mechanism for a privileged process to "get through" to another process, typically to ask it to terminate.

3.2.28.3 Outputs

The Kernel primitive K_signal shall return the following information.

ec: error conditions

The Kernel primitive K_signal shall detect and return the following exception (error) conditions.

EX: process does not have the privilege to issue the call
EX: either the recipient does not exist or is inaccessible due to security or integrity considerations
EX: recipient pseudo interrupt level too high

3.2.29 Kernel Function: K spawn

3.2.29.1 Inputs

The Kernel primitive K_spawn shall take the following parameters.

immSeid: rendezvous name or segment seid of the intermediary process segment
arg: segment designator specifying argument segment

3.2.29.2 Processing

The Kernel primitive K_spawn shall combine the functions of K_fork and K_invoke into one operation. The K_spawn primitive permits process creation without the cost of copying the parent process image to the child process. The effect of K_spawn shall be to create a new process and to force the effect of a K_invoke call upon the newly created process. The parent process may therefore completely specify the contents of the child process image.

The parameters to K_spawn shall be the same as the parameters to the K_invoke primitive. These parameters shall be used to determine the effect of the K_invoke call forced upon the child process. (See K_invoke above for a discussion of this primitive.) The full semantics of K_invoke shall be implemented. Hence, a child process may acquire more privilege than the parent and may operate in a different discretionary access domain.

3.2.29.3 Outputs

The Kernel primitive K_spawn shall return the following information.

childSeid: SEID process identifier of the child process (returned to parent, child process is returned the process SEID of the parent process)
ownSeid: process SEID (parent gets parentSeid, child gets childSeid)
ec: error conditions

The Kernel primitive K_spawn shall detect and return the following exception (error) conditions.

EX: unable to create new process (See note in "K_fork" above.)
EX: invalid argument segment designator
EX: immediary segment does not exist or is inaccessible due to security or integrity considerations
EX: discretionary access violation on intermediary segment
EX: argument segment is not writable
EX: argument segment sharable
EX: argument segment does not fit in space allowed
EX: intermediary segment does not fit in space allowed

3.2.30 Kernel Function: K special function

3.2.30.1 Inputs

The Kernel primitive K_special_function shall take the following parameters.

fn: special function requested
parm.domain: domain of parameter block
parm.idSpace: space of parameter block
parm.vAddr: address of parameter block
parm.size: size of parameter block (may be omitted and default to a standard size)

3.2.30.2 Processing

The K_special_function primitive shall provide a convenient entry point to request infrequently executed functions. At the discretion of the development contractor, subject to Government approval, this primitive may be omitted, and explict calls provided for each function required. The major functions assigned to K_special_function are:

a. shutdown processing

Processes requesting shutdown or to set the system high security and integrity levels shall have at least OPERATOR integrity level. The virtual address of the parameter block shall be checked. The entire parameter block shall lie within one segment.

3.2.30.3 Outputs

The Kernel primitive K_special_function shall return the following information.

ec: error conditions

The Kernel primitive K_special_function shall detect and return the following exception (error) conditions.

EX: insufficient integrity level for function requested
EX: invalid parameter block

3.2.31 Kernel Function: K unlink

3.2.31.1 Inputs

The Kernel primitive K_unlink shall take the following parameters.

fSeid: SEID of file to be unlinked

3.2.31.2 Processing

K_unlink shall decrement the file reference count of the specified file. When the file reference count has decreased to zero, the file shall be deleted after any processes having it open either close it or terminate. If the reference count is zero but the file has not been deleted due to processes having it open, it cannot be opened. The security and integrity checking shall be as if the file were being opened for reading and writing. No discretionary access checking shall be done by the Kernel, allowing processes privileged to use this primitive to perform whatever checking they chose to.

The use of K_unlink may be optionally restricted to processes privileged to issue the call. This option should be exercised when the UNIX Emulator (or similar subsystems including higher level directories) will be included in the system. The unhindered use of K_unlink can harm the integrity of the UNIX directory structure created by the Emulator. In cases in which no Emulator will be included, the restriction on the use of K_unlink may be lifted.

The Directory Manager of the UNIX Emulator [Emulator 78] should perform additional access checks in the emulation of the UNIX unlink(II) call. In particular, the user should have write access to the directory from which the name is being deleted.

3.2.31.3 Outputs

The Kernel primitive K_unlink shall return the following information.

ec: error conditions

The Kernel primitive K_unlink shall detect and return the following exception (error) conditions.

EX: insufficient privilege (optional, see above)
EX: object does not exist or is inaccessible due to security or integrity considerations
EX: object was not a file (devices cannot be unlinked, their SEIDs are permanently assigned)
EX: object resides on a read-only file system

3.2.32 Kernel Function: K unmount

3.2.32.1 Inputs

The Kernel primitive K_unmount shall take the following parameters.

devSeid: device SEID of file system to be dismounted

3.2.32.2 Processing

The Kernel primitive K_unmount shall logically unmount the file system specified by the device SEID. To unmount a file system the following checks must be satisfied:

a. the process must have the privilege to issue the call

b. the user must own the device

c. the device must have file system mounted on it

d. the user must have write permission to the extent

e. the extent must be quiescent (no open files)

3.2.32.3 Outputs

The Kernel primitive K_unmount shall return the following information.

ec: error conditions

The Kernel primitive K_unmount shall detect and return the following exception (error) conditions.

EX: insufficient privilege
EX: device not mounted or is inaccessible due to security or integrity considerations
EX: process owner (user ID) is not the same as the device owner (user ID)
EX: process does not have write discretionary premissions to the device
EX: files still open on the device

3.2.33 Kernel Function: K walk process table

3.2.33.1 Inputs

The Kernel primitive K_walk_process_table shall take the following parameters.

n: global process table entry requested

3.2.33.2 Processing

The K_walk_process_table primitive shall be a means for privileged software to obtain the SEIDs of active processes. The primitive shall return the SEID of the process which occupies slot n of the global process table. This SEID can then be used in K_get_level or K_get_process_status calls. The call shall fail if the process does not posess the privilege to issue it, or if n is not a valid index number for the process table.

3.2.33.3 Outputs

The Kernel primitive K_walk_process_table shall return the following information.

ec:error conditions
rSeid: SEID of the n-th entry in the process table

The Kernel primitive K_walk_process_table shall detect and return the following exception (error) conditions.

EX: not privileged to issue call
EX: n is not a valid process table index

3.2.34 Kernel Function: K write block

3.2.34.1 Inputs

The Kernel primitive K_write_block shall take the following parameters.

od: open object descriptor
blockNo: logical block number of the file at which to start writing
duFile.vl.domain: domain in which data to write resides
duFile.vl.vAddr: address at which data starts
duFile.vl.idSpace: space of data area
duFile.size: length (in bytes) of data to write
id: asynchronous call identifier

3.2.34.2 Processing

K_write_block shall move the data in the area described by the parameter block into the specified block(s) of the file designated by od. Security and discretionary access conditions shall be checked at time of K_open on the file. If the file was not opened for writing (or for reading and writing) an exception shall be returned. Other exceptions shall include bad parameter values and physical I/O errors.

The Kernel shall not maintain information about where in the file the last I/O was directed. Thus, it is the responsibility of software using the Kernel to keep such information.

Limited numbers of asynchronous K_write_block requests may be made. If the the asynchronous call identifier field is non-null, K_write_block may return before completion of the requested operation. At the completion of the requested operation, the Kernel shall transmit an message to the requesting process containing (at least): the asynchronous call identifier, the number of bytes written, and other device dependent status information.

3.2.34.3 Outputs

The Kernel primitive K_write_block shall return the following information.

ec: error conditions

The Kernel primitive K_write_block shall detect and return the following exception (error) conditions.

EX: parameter block invalid, or segment designated is not readable
EX: open descriptor invalid
EX: open descriptor not opened for writing
EX: attempt to write to a blocked terminal
EX: size of block does not match device block size or is outside allowable limits for device
EX: block number out of range
EX: size of data area would result in a block number out of range.

3.2.35 Kernel Resource Manager

`. _ KSOS Security Kernel shall include a system resource manager that is logically separate from the physical scheduling mechanisms of the Kernel. The Kernel Resource Manager may either be made part of the Kernel or may exist as an external process. The Kernel Resource Manager shall perform higher level management of:

a. processor or CPU resources

b. physical memory

c. process segment swapping space

The Kernel Resource Manager shall attempt to maximize the utilization of all system-wide resources. The Resource Manager shall attempt to guarantee fair and equitable service to all processes running in the KSOS environment. Denial of service conditions shall be avoided whenever possible. Deadlock conditions shall also be avoided.

The operation of the Kernel Resource Manager requires a tightly coupled interface with the rest of the Kernel. If the Kernel Resource Manager is implemented as an external process, care shall be taken to make the interface as efficient and secure as possible.

3.2.36 Secure Terminal Interface

To assure non-spoofable communications between the user terminal and KSOS trusted software, the Security Kernel shall provide a secure terminal interface. This function shall operate as follows. One of the characters which can be typed by the user shall be reserved for the Kernel attention character. When the user transmits this character, the Kernel shall suspend all normal input/output between the terminal and the users' processes. The Kernel shall send an IPC message to the Secure Initiator [NKSR 78] notifying this process that a user at the specified terminal has requested a special secure service. Further communication with the terminal shall only be permitted to processes permitted to open the terminal using a SEID specifying secure terminal communications. Normal communications shall resume after a thaw command has been issued by an appropriately privileged process, through the K_secure_terminal_lock primitive.

3.2.37 Audit Messages

Standard DoD military security policy requires a record of security related operations which may be subject to audit. The KSOS Kernel shall transmit audit messages to the secure Audit Capture Process [NKSR 78]. The Security Kernel shall generate audit messages for (at least) the following security related events.

a. file creation

b. file deletion

c. any operation that changes the file owner, security level, integrity level, or process privileges associated with process image files

d. available resource exhaustion conditions for the resources:

   1. available processes

   2. file space

   3. IPC message buffer space.

e. any access attempt denied because of security or integrity violations

Audit messages may be generated for other security related events to be determined by the development contractor subject to Government review and approval. Kernel-level audit functions shall be capable of being selectively disabled during system generation at the discretion of the System Security Officer. The exact format of the audit messages and the mechanism by which they are transmitted shall be left to the development contractor subject to Government approval.

3.2.38 Kernel Trap Management

3.2.38.1 Inputs

The inputs to the Security Kernel trap management function shall be the various processor exception conditions and device interrupts. On the PDP-11/70, these conditions are vectored to specific memory locations which shall be accessible only from kernel domain software.

3.2.38.2 Processing

The Security Kernel shall dispatch device interrupts to the appropriate peripheral device handler. In the PDP-11/70 implementation of KSOS, device interrupt handling shall be limited to the Security Kernel. In other computer architectures which can guarantee the protection and security of user level device handling, These functions could be located outside the perimeter of the Security Kernel.

Certain processor exception conditions shall be reflected to the supervisor domain of a user process, e.g., the UNIX Emulator. The processor exception conditions reflected outwards shall be those traps caused by the software executing outside the domain of the Kernel. The PDP-11/70 version of KSOS, the following traps shall be reflected to supervisor domain software by means of the pseudo-interrupt mechanism:

a. Illegal or reserved instruction exceptions. Illegal instruction codes. Reserved instructions executed in user or supervisor domain.

b. Breakpoint traps. Execution of the BPT instruction. Trace traps.

c. IOT instruction traps. (Causes UNIX abort.)

d. TRAP instruction. (Causes entry to UNIX emulator dispatch.)

e. Floating point errors. Illegal floating point operation codes. Floating to integer conversion errors. Floating underflow and overflow. Floating undefined variable. Maintenance trap.

f. Memory management traps. Non-resident segment trap. Segment length abort. Read only access violation. Memory management trap.

Certain trap conditions represent either system hardware failures, or serious problems internal to the Kernel. These trap conditions shall cause an unplanned system shutdown. Where possible, the system should communicate the cause of the unplanned shutdown either by sending information to the Audit Captuie Process or by printing a message on the system console.

3.2.39 Pseudo Interrupts

The Kernel shall provide a pseudo interrupt facility to be used by the IPC mechanism and for trap reflection and timer interrupts. The mechanism for the PDP-11/70 implementation shall set the program counter and processor status word to values retrieved from fixed locations in the supervisor domain. The processor status word shall be masked before actually being used to prevent the user from specifying the kernel mode. Processes shall be able to inhibit pseudo interrupts through the appropriate settings of parameters in the type dependent information for the process. The K_interrupt_return call shall be the mechanism for resuming normal processing after a pseudo interrupt. The call shall restore the state of the process from the values contained in the pseudo interrupt vector. These locations may be modified by the software involved in interrupt processing. The Kernel shall prevent the process from increasing its accessible domains by masking the processor status word before restoring it.

3.2.40 Special Requirements

Because the Security Kernel CPCI is the heart of the system's security, it shall be produced with extraordinary precision and thoroughness. The System Specification (Type A) [A-Specs 78] describes the design and implementation standards to be followed in the Security Kernel CPCI. The Security Kernel shall be designed and implemented in a manner which is amenable to eventual formal proof of the security properties of the system.

The Security Kernel shall be designed to facilitate its modification and enhancement. It should be recognized that any modifications to the Security Kernel may invalidate all or part of the proofs of system security. Thus, modification efforts must be carefully controlled. No field maintenance of the Security Kernel shall be permitted.

3.2.40.1 Human Performance

This paragraph is not applicable to this specification.

3.2.40.2 Government Furnished Property List

The Security Kernel CPCI shall be implemented in a Government-approved language. The compiler for this language may be furnished by the Government.

3.3 Adaptation

The Security Kernel shall be capable of adapting to any of the hardware configurations shown in the System Specification [A-Specs 78]. The creation of a Security Kernel for a particular configuration shall be an automated process designed to be performed by personnel without extensive computer knowledge. The mechanism shall employ checksums or similar techniques to assure the integrity of the inputs to the system generation process. The Security Kernel shall operate correctly in a configuration that is reduced from the one for which it was generated. This KSOS System Generation mechanism is discussed in the Non-Kernel Security Related Software CPCI Specification [NKSR 78].

3.3.1 General Environment

The Kernel and the Non-Kernel Security Related Software shall establish and enforce a maximum security level for the system as a whole and for each peripheral device and disk extent. In specifying the maximum level for peripheral devices, the System Security Officer should consider the physical location of the device, the protection of any connecting communication lines, and the personnel which would have access to the device. Individual sites may be governed by specific policies in this area.

3.3.2 System Parameters

The size of all system tables and arrays shall be parameterized to allow them to be changed easily. All manifest constants shall also be parameterized.

3.3.3 System Capacities

The Security Kernel shall be capable of supporting the maximum hardware configuration defined in the System Specifications. All terminals in that configuration shall be capable of being simultaneously on-line. The Security Kernel shall be capable of supporting at least fifty (50) active processes.

4. QUALITY ASSURANCE PROVISIONS

4.1 Introduction

The underlying philosophy of the overall KSOS quality assurance is to provide a convincing demonstration of the security and completeness of the system. Testing and quality assurance are a complement to the formal verification aspects of KSOS. All testing on KSOS shall be conducted at the development contractor's facility.

KSOS shall be subject to the technical reviews and audit procedures of MIL-STD-1521A, Appendices A-F. The following technical reviews and audits will be conducted:

a. System Requirements Review (Appendix A)

b. System Design Review (Appendix B)

c. Preliminary Design Review (Appendix C)

d. Critical Design Review (Appendix D)

e. Functional Configuration Audit (Appendix E)

f. Physical Configuration Audit (Appendix F)

The Critical Design Review shall include a review of any mathematical proofs showing that the design meets the requirements of the Government approved DoD security model.

The Functional Configuration Audit shall be the vehicle for the formal review of the design versus the mathematical description. The Physical Configuration Audit shall be the vehicle for the formal review of the "as built" computer programs versus both their design and their supporting documentation. In addition, the Physical Configuration Audit shall verify that the implementation requirements of the Type A System Specifications have been met. Both audits shall include review of any relevant mathematical proofs of correctness.

4.1.1 Category I Testing

The demonstration that the requirements of Section 3 have been met shall consist of a combination of formal inspection of the CPCI and tests which demonstrate that the CPCI meets its functional requirements. The tests on the Security Kernel shall exercise each of the options for each primitive. To the extent feasible, the tests shall attempt to exercise combinations of parameters to the Kernel calls. Test sequences shall include cases which are expected to fail, for example tests which deliberately attempt to violate security. All the exception conditions defined in Section 3. shall be so tested.

4.1.2 Category II Testing

The KSOS system shall be subject to an overall system test (Category II) in accordance with the System Specification (Type A) and the development contract.

This Category II test shall demonstrate that the three CPCI's which make up the KSOS system function correctly together. The Category II testing shall attempt to represent a typical user load including at least ten (10) on-line terminals and a network connection. To the extent feasible, the Category II testing shall also be automated through the use of shell scripts, and (optionally) another computer system emulating the terminal load.

Table I - Quality Assurance

Assurance techniques:

NA	Not Applicable
V	Verification
F	Formal Specification
T	Testing
D	Demonstration
A	Analysis
I	Inspection

Applicable test types:

1	Module level testing
2	Integration testing
3	Acceptance testing

Requirement	Assurance Techniques							Test types		
	NA	V	F	T	D	A	I	1	2	3
3.1.1.1.				x	x	x	x	x	x	x
3.1.1.2.1.1.			x	x	x	x	x	x	x	x
3.1.1.2.1.2.			x	x	x	x	x	x	x	x
3.1.1.2.1.3.			x	x	x	x	x	x	x	x
3.1.1.2.1.4.			x	x	x	x	x	x	x	x
3.1.1.2.1.5.			x	x	x	x	x	x	x	x
3.1.1.2.2.				x	x	x	x	x	x	x
3.1.1.2.3.	x									
3.1.1.2.4.				x	x	x	x	x	x	x
3.1.1.2.5.			x	x	x	x	x	x	x	x
3.1.1.2.6.				x	x	x	x	x	x	x
3.2.1. - 3.2.34		*	x	x	x	x	x	x	x	x
3.2.35.			x	x	x	x	x	x	x	x
3.2.36.				x	x	x	x	x	x	x
3.2.37.			x	x	x	x	x	x	x	x
3.2.38.			x	x	x	x	x	x	x	x
3.2.39.			x	x	x	x	x	x	x	x
3.2.40.1.	x									
3.2.40.2.	x									
3.3.						x	x			
3.3.1.						x	x			
3.3.2.							x			
3.3.3.				x	x			x	x	x

* Selected modules will be shall be formally verified on a time and materials basis.

5. PREPARATION FOR DELIVERY

This section is not applicable to this specification.

6. NOTES

This section is not applicable to this specification.

10. APPENDIX

10. APPENDIX

This appendix presents the formal specifications for the KSOS Security Kernel. The specifications describe the interface that the Kernel shall present to its users at the Kernel call level. These formal specifications and the textual descriptions of the Kernel call interface found in Section 3 are intended to be be used side-by-side, each serving to illuminate the other. The formal specifications are written in the SPECIAL (SPECIfication and Assertion Language) language described by [Roubine and Robinson 77].

For the purposes of specification, the Kernel has been decomposed into several modules which are viewed as being hierarchically ordered. A convention has been followed that each module is referred to by a three-letter mnemonic name derived from its function.

Each of the specification modules represents an incremental abstract machine, built upon the abstract machines that come below it in the hierarchy. A module contains descriptions of the data types which it manipulates, and functions which represent the state variables (V-functions) and primitive operations (O-functions) of the abstract machine. The functions of a module may be thought of as being the mechanism of that module's abstract machine.

Not all the functions of a module are necessarily "visible to" (i.e. available for use by) other modules. Furthermore, it is a strict rule of the design methodology employed that no function at a given level in the hierarchy is ever visible below that level.

The modules contain varying amounts of mechanism. This is dictated by the extent to which the Kernel calls they mechanize alter the state of the Kernel. A ground rule followed in writing these specifications has been to specify nothing which is not somehow visible at the Kernel call interface.

All the Kernel calls, data types, and data structures which are visible at the Kernel call interface have been collected, for ease of reference into the topmost layer of the hierarchy ("KER"). The effects of any Kernel call may be determined from the specifications by finding the entry for that call in KER, and then following the flow of implication through the hierarchy.

Another useful way in which the specifications may be studied is to determine the mechanisms and structures supporting a particular Kernel feature. This is best done by examination of the module(s) which specify the feature.

SRI International has developed and maintains tools which can verify the internal consistency of a set of formal specifications written in SPECIAL. These specifications have been processed through these tools and have been judged by the tools to be syntactically correct and semantically consistent. In the future these specifications will be processed by other tools to verify that they do indeed provide a system consistent with the mathematical security model.

```
$("     MODULE:         ker.specs (version 4.8)
        CONTENTS:       Kernel Calls
        TYPE:           SPECIAL.specifications
        LAST CHANGED:   12/23/80, 16:51:22
  ")


MODULE ker


    TYPES

    $(from mac)
vAddrType: {0 .. MACmaxVAddr};

    $(from smx)
nonDisType: STRUCT_OF(
                INTEGER securityLevel; SET_OF securityCat securityCatS;
                INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
daType: SET_OF daMode;
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
tiiStruct: STRUCT_OF(nonDisType nd;
                modeStruct ms; INTEGER owner, group; SET_OF privType priv);

    $(FROM pvm)
virtualLocation:
  STRUCT_OF(domainType domain; spaceType idSpace; vAddrType vAddr);
globalData:
  STRUCT_OF(BOOLEAN sharable, swappable, sticky, memAdvise, executable);
statusStruct:
  STRUCT_OF(globalData gl; direction growth; INTEGER size; seid iseid;
                  segDes udes; virtualLocation vloc; daType da; BOOLEAN mapped);
pBlock: STRUCT_OF(virtualLocation vloc; INTEGER size);

    $(FROM fca)
fileStatus: STRUCT_OF(INTEGER nBlocks, linkCount, timeLastMod; seid subtype;
                      BOOLEAN openAtCrash);
asyncId: CHAR;
ioStatus: STRUCT_OF(INTEGER devIndep, devDep);
fileNsp:  {129 .. 255};         $(nsp values for file systems)

    $(from pro)
piLevelType: {PROminPiLevel .. PROmaxPiLevel}; $(pseudo interrupt level range)
piEntryType: STRUCT_OF(piLevelType oldPil;
                       piLevelType oldPil;
                           INTEGER oldPc;
                           INTEGER oldPs;
                           INTEGER parameter;
                           INTEGER newPc;
                           INTEGER newPs);
piVectorType: {VECTOR_OF piEntryType piv | LENGTH(piv) = PROmaxPiLevel -PROminPiLevel};
ipcqType: {VECTOR_OF ipcMessageType zz | LENGTH(zz) <= IPCmaxMessageCount};
ipcTextType: {VECTOR_OF CHAR vc | LENGTH(vc) = IPCmaxMessageLength};
ipcMessageType: STRUCT_OF(seid sender; ipcTextType text);
processStatusType: STRUCT_OF(seid self;
                                 seid parent;
```

```
                            INTEGER family;
                            INTEGER realUser;
                            INTEGER realGroup;
                            INTEGER pc;          $(program counter)
                            INTEGER ps;          $(processor status)
                        piLevelType pil;
                       piVectorType piv;
                           ipcqType ipcq;
                            INTEGER advPrio;     $(advisory priority)
                            INTEGER timerAlarm;  $(one*zero crossing => pi)
                            INTEGER supervisorTiming;
                            INTEGER userTiming;
                            BOOLEAN timTog);     $(timer toggle TRUE is ON)


    EXTERNALREFS

    FROM mac:
INTEGER MACmaxVAddr;

    FROM smx:
seid: DESIGNATOR;
        privType: {
                             privFileUpdateStatus,  privLink,
                             privLockSeg,           privModifyPriv,
                             privMount,             privSetLevel,
                             privStickySeg,         privSetPath,
                             privViolSimpSecurity,  privViolStarSecurity,
                             privViolSimpIntegrity, privViolStarIntegrity,
                             privViolDiscrAccess,   privRealizeExecPermission,
                             privSignal,            privWalkPTable,
                             privHalt,              privKernelCall,
                             privViolCompartments,  privSetComm,
                             privImmigrate
                             };

daMode: {daRead, daWrite, daExecute};
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;
domainType: {nullDomain, kernelDomain, supervisorDomain, userDomain};

    FROM pvm:
segDes: DESIGNATOR;
spaceType: {nullSpace,iSpace, dSpace};
direction: {up, down};
OVFUN PVMbuild(seid pSeid; statusStruct ss; modeStruct ms)
  *> STRUCT_OF(seid segSeid; segDes segd) result;
OFUN PVMdestroy(seid pSeid; segDes segd);
VFUN PVMgetSegmentStatus(seid pSeid, segSeid; segDes segd) *> statusStruct st;
OFUN PVMsetSegmentStatus(seid pSeid, segSeid; globalData gl);
OFUN PVMremap(seid pSeid; segDes in; virtualLocation vl; daType d;
          BOOLEAN vlFlg, daFlg; segDes out; INTEGER outSize;
          BOOLEAN osFlg);
OVFUN PVMrendezvous(seid pSeid, segSeid; virtualLocation vl; daType d)
  *> segDes segd;

```

```
    FROM fca:
openDescriptor: DESIGNATOR;
openModes: {omRead, omWrite, omExclusiveRead, omExclusiveWrite};
IOfunction: {rewind, etc};
OFUN FCAclose(seid pSeid; openDescriptor od);
OVFUN FCAcreate(seid pSeid, nspSeid; SET_OF openModes mode; modeStruct d;
   openDescriptor stCap)
  *> STRUCT_OF(seid fSeid; openDescriptor od) result;
OVFUN FCAopen(seid pSeid, oSeid; SET_OF openModes om; openDescriptor stCap)
  *> openDescriptor od;
VFUN FCAgetFileStatus(seid pSeid, fSeid) *> fileStatus fst;
OFUN FCAsetFileStatus(seid pSeid, fSeid;  ileStatus newfst);

    FROM fmi:
OFUN FCAlink(seid pSeid, fSeid);
OVFUN FCAmount(seid pSeid, dev, root; BOOLEAN readOnly) *> fileNsp newNsp;
OFUN FCAterminalLock(seid pSeid, devSeid);
OFUN FCAunlink(seid pSeid, fSeid);
OFUN FCAunmount(seid pSeid; fileNsp fsNsp);


    FROM pro:
INTEGER IPCmaxMessageCount;
INTEGER IPCmaxMessageLength;
INTEGER PROminPiLevel;
INTEGER PROmaxPiLevel;
OVFUN PROfork(seid pSeid) *> seid childSeid;
VFUN PROgetProcessStatus(seid pSeid, getSeid) *> processStatusType ps;
OFUN PROinterruptReturn(seid pSeid);
OFUN PROinvoke(seid pSeid, immSeid; segDes arg);
OFUN PROnap(seid pSeid; INTEGER timeOut);
OFUN PROpost(seid pSeid, receiver; BOOLEAN pseudoInt; ipcTextType msg);
OVFUN PROreceive(seid pSeid; INTEGER timeOut) *> ipcMessageType msg;
OFUN PROreleaseProcess(seid pSeid, rSeid);
OFUN PROsetProcessStatus(seid pSeid, procSeid; processStatusType ps);
OFUN PROsignal(seid sender, receiver; ipcTextType signalMsg);
OVFUN PROspawn(seid pSeid, immSeid; segDes arg) *> seid childSeid;
VFUN PROwalkProcessTable(seid pSeid; INTEGER n) *> seid rSeid;

    FROM lev:
VFUN LEVgetObjectLevel(seid pSeid, objSeid) *> tiiStruct level;
OFUN LEVsetObjectLevel(seid pSeid, objSeid; tiiStruct level);

    FROM spf:
SPFfunctionType: {syncSPF, immSegSPF, sysHaltSPF, levelSetSPF};

    FROM pbl:
OFUN PBLdeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
                       pBlock arguments, status; asyncId id);
OVFUN PBLreadBlock(seid pSeid; openDescriptor od; INTEGER blockNo;
                   pBlock duFile; asyncId as)
  *> STRUCT_OF(INTEGER bytesRead; ioStatus errst) result;
OFUN PBLspecialFunction(seid pSeid; SPFfunctionType fn; pBlock parm);
OVFUN PBLwriteBlock(seid pSeid; openDescriptor od; INTEGER blockNo;
                    pBlock duFile; asyncId as)
  *> ioStatus ios;
```

```


    FUNCTIONS

$("Note regarding implementation:  The implicit parameter pSeid is
implemented by passing a null seid to the kernel interface.  The null
seid is converted upon entering the interface to the seid of the
calling process.  As a consequence, for a process to learn its own
seid, it suffices to call K_get_Process_Status with a null seid.")

$(Visible Kernel Functions in Alphabetical Order)

OVFUN K_build_segment(statusStruct ss; modeStruct ms)
        [seid pSeid] '> STRUCT_OF(seid segSeid; segDes segd) result;
                                                           $(K_build_segment)
  EXCEPTIONS
    EXCEPTIONS_OF PVMbuild(pSeid, ss, ms);
  EFFECTS
    result = EFFECTS_OF PVMbuild(pSeid, ss, ms);

OFUN K_close(openDescriptor od)[seid pSeid];                       $(K_close)
  EXCEPTIONS
    EXCEPTIONS_OF FCAclose(pSeid, od);
  EFFECTS
    EFFECTS_OF FCAclose(pSeid, od);

OVFUN K_create(seid nspSeid; SET_OF openModes mode; modeStruct d;
        openDescriptor stCap)[seid pSeid]
        '> STRUCT_OF(seid fSeid; openDescriptor od) return;      $(K_create)
  EXCEPTIONS
    EXCEPTIONS_OF FCAcreate(pSeid, nspSeid, mode,d, stCap);
  EFFECTS
    return = EFFECTS_OF FCAcreate(pSeid, nspSeid, mode,d, stCap);

OFUN K_device_function(openDescriptor od; IOfunction f;
                       pBlock arguments, status; asyncId id)
       [seid pSeid];                                     $(K_device_function)
  EXCEPTIONS
    EXCEPTIONS_OF PBLdeviceFunction(pSeid, od, f, arguments, status, id);
  EFFECTS
    EFFECTS_OF PBLdeviceFunction(pSeid, od, f, arguments, status, id);

OVFUN K_fork()[seid pSeid] '> seid childSeid;                       $(K_fork)
  EXCEPTIONS
    EXCEPTIONS_OF PROfork(pSeid);
  EFFECTS
    childSeid = EFFECTS_OF PROfork(pSeid);
    $("ChildSeid is returned to the (original) process; pSeid (parent seid)
      is returned to the newly created child")

VFUN K_get_file_status(seid fSeid)[seid pSeid] '> fileStatus fst;
                                                         $(K_get_file_status)
  EXCEPTIONS
    EXCEPTIONS_OF FCAgetFileStatus(pSeid, fSeid);
  DERIVATION
    FCAgetFileStatus(pSeid, fSeid);
```

```

VFUN K_get_object_level(seid objSeid)[seid pSeid] -> tiiStruct otii;
                                                        $(K_get_object_level)
  EXCEPTIONS
    EXCEPTIONS_OF LEVgetObjectLevel(pSeid, objSeid);
  DERIVATION
    LEVgetObjectLevel(pSeid, objSeid);

VFUN K_get_process_status(seid getSeid)[seid pSeid] -> processStatusType ps;
                                                        $(K_get_process_status)
  EXCEPTIONS
    EXCEPTIONS_OF PROgetProcessStatus(pSeid, getSeid);
  DERIVATION
    PROgetProcessStatus(pSeid, getSeid);

VFUN K_get_segment_status(seid segSeid; segDes segd)[seid pSeid] ->
                                                        statusStruct st;
                                                        $(K_get_segment_status)
  EXCEPTIONS
    EXCEPTIONS_OF PVMgetSegmentStatus(pSeid, segSeid,segd);
  DERIVATION
    PVMgetSegmentStatus(pSeid, segSeid,segd);

OFUN K_interrupt_return()[seid pSeid];                  $(K_interrupt_return)
  EXCEPTIONS
    EXCEPTIONS_OF PROinterruptReturn(pSeid);
  EFFECTS
    EFFECTS_OF PROinterruptReturn(pSeid);

OFUN K_invoke(seid immSeid; segDes arg)[seid pSeid];              $(K_invoke)
  EXCEPTIONS
    EXCEPTIONS_OF PROinvoke(pSeid, immSeid, arg);
  EFFECTS
    EFFECTS_OF PROinvoke(pSeid, immSeid, arg);

OFUN K_link(seid fSeid)[seid pSeid];                                $(K_link)
  EXCEPTIONS
    EXCEPTIONS_OF FCAlink(pSeid, fSeid);
  EFFECTS
    EFFECTS_OF FCAlink(pSeid, fSeid);

OVFUN K_mount(seid dev; BOOLEAN readOnly)[seid pSeid]        $(K_mount)
                         -> fileNsp newNsp;
  EXCEPTIONS
    EXCEPTIONS_OF FCAmount(pSeid, dev, readOnly);
  EFFECTS
    newNsp = EFFECTS_OF FCAmount(pSeid, dev, readOnly);

OFUN K_nap(INTEGER timeOut)[seid pSeid];                             $(K_nap)
  EXCEPTIONS
    EXCEPTIONS_OF PROnap(pSeid, timeOut);
  EFFECTS
    EFFECTS_OF PROnap(pSeid, timeOut);

OVFUN K_open(seid oSeid; SET_OF openModes om; openDescriptor stCap)
          [seid pSeid] -> openDescriptor od;                        $(K_open)
  EXCEPTIONS
    EXCEPTIONS_OF FCAopen(pSeid, oSeid, om, stCap);
  EFFECTS
    od = EFFECTS_OF FCAopen(pSeid, oSeid, om, stCap);

OFUN K_post(seid receiver; BOOLEAN pseudoInt; ipcTextType msg)[seid pSeid];
                                                                 $(K_post)
  EXCEPTIONS
    EXCEPTIONS_OF PROpost(pSeid, receiver, pseudoInt, msg);
  EFFECTS
    EFFECTS_OF PROpost(pSeid, receiver, pseudoInt, msg);

OVFUN K_read_block(openDescriptor od; INTEGER blockNo; pBlock duFile;
                   asyncId as)[seid pSeid]
  *> STRUCT_OF(INTEGER bytesRead; ioStatus errst) result;     $(K_read_block)
  EXCEPTIONS
    EXCEPTIONS_OF PBLreadBlock(pSeid, od, blockNo, duFile, as);
  EFFECTS
    result = EFFECTS_OF PBLreadBlock(pSeid, od, blockNo, duFile, as);

OVFUN K_receive(INTEGER timeOut)[seid pSeid] *> ipcMessageType msg;
                                                              $(K_receive)
  EXCEPTIONS
    EXCEPTIONS_OF PROreceive(pSeid, timeOut);
  EFFECTS
    msg = EFFECTS_OF PROreceive(pSeid, timeOut);

OFUN K_release_process(seid rSeid)[seid pSeid];         $(K_release_process)
  EXCEPTIONS
    EXCEPTIONS_OF PROreleaseProcess(pSeid, rSeid);
  EFFECTS
    EFFECTS_OF PROreleaseProcess(pSeid, rSeid);

OFUN K_release_segment(segDes segd)[seid pSeid];        $(K_release_segment)
  EXCEPTIONS
    EXCEPTIONS_OF PVMdestroy(pSeid, segd);
  EFFECTS
    EFFECTS_OF PVMdestroy(pSeid, segd);

OFUN K_remap(segDes in; virtualLocation vl; daType d;  BOOLEAN vlFlg, daFlg;
             segDes out; INTEGER outSize; BOOLEAN osFlg)[seid pSeid];
                                                                $(K_remap)
  EXCEPTIONS
    EXCEPTIONS_OF
      PVMremap(pSeid, in, vl, d, vlFlg, daFlg, out, outSize, osFlg);
  EFFECTS
    EFFECTS_OF
      PVMremap(pSeid, in, vl, d, vlFlg, daFlg, out, outSize, osFlg);

OVFUN K_rendezvous_segment(seid segSeid; virtualLocation vl; daType d)
                           [seid pSeid]
      *> segDes segd;                               $(K_rendezvous_segment)
  EXCEPTIONS
    EXCEPTIONS_OF PVMrendezvous(pSeid, segSeid, vl, d);
  EFFECTS
    segd = EFFECTS_OF PVMrendezvous(pSeid, segSeid, vl, d);
```

```

OFUN K_secure_terminal_lock(seid devSeid)[seid pSeid];
                                                   $(K_secure_terminal_lock)
  EXCEPTIONS
    EXCEPTIONS_OF FCAterminalLock(pSeid, devSeid);
  EFFECTS
    EFFECTS_OF FCAterminalLock(pSeid, devSeid);

OFUN K_set_file_status(seid fSeid; fileStatus fst)[seid pSeid];
                                                        $(K_set_file_status)
  EXCEPTIONS
    EXCEPTIONS_OF FCAsetFileStatus(pSeid, fSeid, fst);
  EFFECTS
    EFFECTS_OF FCAsetFileStatus(pSeid, fSeid, fst);

OFUN K_set_object_level(seid objSeid; tiiStruct level)[seid pSeid];
                                                       $(K_set_object_level)
  EXCEPTIONS
    EXCEPTIONS_OF LEVsetObjectLevel(pSeid, objSeid, level);
  EFFECTS
    EFFECTS_OF LEVsetObjectLevel(pSeid, objSeid, level);

OFUN K_set_process_status(seid procSeid; processStatusType ps)[seid pSeid];
                                                     $(K_set_process_status)
  EXCEPTIONS
    EXCEPTIONS_OF PROsetProcessStatus(pSeid, procSeid, ps);
  EFFECTS
    EFFECTS_OF PROsetProcessStatus(pSeid, procSeid, ps);

OFUN K_set_segment_status(seid segSeid; globalData gl)[seid pSeid];
                                                     $(K_set_segment_status)
  EXCEPTIONS
    EXCEPTIONS_OF PVMsetSegmentStatus(pSeid, segSeid, gl);
  EFFECTS
    EFFECTS_OF PVMsetSegmentStatus(pSeid, segSeid, gl);

OFUN K_signal(seid procSeid; ipcTextType sigMsg)[seid pSeid];        $(K_signal)
  EXCEPTIONS
    EXCEPTIONS_OF PROsignal(pSeid, procSeid, sigMsg);
  EFFECTS
    EFFECTS_OF PROsignal(pSeid, procSeid, sigMsg);

OVFUN K_spawn(seid immSeid; segDes arg)[seid pSeid] -> seid childSeid;
                                                                 $(K_spawn)
  EXCEPTIONS
    EXCEPTIONS_OF PROspawn(pSeid, immSeid, arg);
  EFFECTS
    childSeid = EFFECTS_OF PROspawn(pSeid, immSeid, arg);

OFUN K_special_function(SPFfunctionType fn; pBlock parm)[seid pSeid];
                                                      $(K_special_function)
  EXCEPTIONS
    EXCEPTIONS_OF PBLspecialFunction(pSeid, fn, parm);
  EFFECTS
    EFFECTS_OF PBLspecialFunction(pSeid, fn, parm);

```

```
OFUN K_unlink(seid fSeid)[seid pSeid];                                    $(K_unlink)
  EXCEPTIONS
    EXCEPTIONS_OF FCAunlink(pSeid, fSeid);
  EFFECTS
    EFFECTS_OF FCAunlink(pSeid, fSeid);

OFUN K_unmount(fileNsp fsNsp)[seid pSeid];                                $(K_unmount)
  EXCEPTIONS
    EXCEPTIONS_OF FCAunmount(pSeid, fsNsp);
  EFFECTS
    EFFECTS_OF FCAunmount(pSeid, fsNsp);

VFUN K_walk_process_table(INTEGER n)[seid pSeid] => seid rSeid;
                                                                 $(K_walk_process_table)
  EXCEPTIONS
    EXCEPTIONS_OF PROwalkProcessTable(pSeid,n);
  DERIVATION
    PROwalkProcessTable(pSeid,n);

OVFUN K_write_block(openDescriptor od; INTEGER blockNo; pBlock duFile;
                    asyncId id)[seid pSeid]
  => ioStatus ios;                                                    $(K_write_block)
  EXCEPTIONS
    EXCEPTIONS_OF PBLwriteBlock(pSeid, od, blockNo, duFile, id);
  EFFECTS
    ios = EFFECTS_OF PBLwriteBlock(pSeid, od, blockNo, duFile, id);


END_MODULE
```

```
$("     MODULE:         ddf.specs (version 2.5)
        CONTENTS:       Device Dependent Functions
        TYPE:           SPECIAL.specifications
        LAST CHANGED:   7/28/80, 09:22:51
")


MODULE ddf

TYPES
deviceType: {RK05, RWP04, RWP05, RWP06, RSW04, TWE16, TM11, TU56, PR11,
             PC11, LP11, IMP11B, LHDH};
deviceStruct: STRUCT_OF(BOOLEAN addressable;
                        INTEGER minRequest, maxRequest, modSize, maxBlockNo);
  $(properties necessary for processing IO requests for devices)

FUNCTIONS

VFUN DDFdeviceData(deviceType d) -> deviceStruct ds;         $(DDFdeviceData)
  $(for a given type of device, defines the IO behavior)
  INITIALLY
    ds = (IF d = RK05
            THEN STRUCT(TRUE, 512, 65534, 512, 203*24-1)
          ELSE IF d INSET {RWP04, RWP05}
            THEN STRUCT(TRUE, 512, 65534, 512, 22*19*411-1)
          ELSE IF d = RWP06
            THEN STRUCT(TRUE, 512, 65534, 512, 22*19*411*2-1)
          ELSE IF d = RSW04 THEN STRUCT(TRUE, 512, 65534, 512, 2048-1)
          ELSE IF d INSET {TWE16, TM11} THEN STRUCT(FALSE, 12, 8191, 1, 0)
          ELSE IF d = TU56 THEN STRUCT(TRUE, 512, 512, 512, 578-1)
          ELSE IF d INSET {PR11, PC11} THEN STRUCT(FALSE, 1, 1, 1, 0)
          ELSE IF d = LP11 THEN STRUCT(FALSE, 1, 132, 1, 0)
          ELSE IF d INSET {IMP11B, LHDH} THEN STRUCT(FALSE, 1, 8191, 1, 0)
          ELSE ?);

END_MODULE
```

```
$("      MODULE:          fca.specs (version 4.16)
         CONTENTS:        File Capabilities
         TYPE:            SPECIAL.specifications
         LAST CHANGED:    12/23/80, 16:34:45
  ")


MODULE fca


$("This module manages all openable objects, i.e., those that are referenced
   through the open table corresponding to a process.  These objects include
   files, devices * both addressable and nonaddressable*, terminals, extents,
   and subtypes.

   Each object is identified by a seid.  Seids for devices, terminals, and
   subtypes are allocated at system generation time.  These objects are
   permanent, and cannot be dynamically allocated and deallocated.
   Seids for files are allocated by this module. Seids for extents are
   allocated when the device is physically mounted.  Physicial mounting
   is not handled at this time ** logical mounting is ** but should be.

   Each process at creation is assigned an open table, in which all the
   open objects of that process are recorded, along with their mode of
   access.  The state of the open table for a process is recorded in the
   values of the V*functions 'FCAopenTableExists(pSeid)', which tells
   whether the open table for the process named by 'pSeid' exists, and
   'FCAopenEntry(pSeid, od)', which gives the seid and open mode for the
   open object of process 'pSeid' named by the open Descriptor ** a
   designator ** 'od.'

   The existence of an openable object is detected by a defined value for
   'FCAfileStatusInfo(fSeid),' where 'fSeid' is the object's seid.  Each
   object's type is ascertained by looking at the nsp part of the seid.
   Depending on the type of the object, certain V*functions hold additional
   information. A description of this information can be found in the
   comment directed at each type of object.")

$(" DEVICES ** there are two kinds of devices, addressable and nonaddressable;
    an addressable device, such as a disk, has two properties: it can be
    accessed via a block number or address; and what is put onto the device
    via a read operation is retrieved by a write when the device is read at
    the same address.  A non*addressable device, such as a tape unit, can be
    viewed as having an infinite stream of input data and producing an
    infinite stream of output data.  Each kind of device has four quantities
    associated with it: a minimum request, a maximum request, a size
    modulus, and a maximum block number.

    For non*addressable devices, the
    size modulus must be 1 and the maximum block number is meaningless.
    An IO request specifies a certain number of characters at a certain
    block number.  The number of characters must be within the range
    defined by the minimum and maximum request quantities for the device,
    and must be a multiple of the size modulus. The block number must be with
    within the range {0 .. maximum block number} for the device.
")
```

```

$(" TERMINALS -- With one exception, terminals are nonaddressable devices
    whose IO requests are limited to small multiples of a single character
    (~ 255).  Terminals however, have a special property.  To enable the
    user to change the security level of his job without changing terminals,
    there is an illusion that a single terminal is represented by a
    multiplicity of device seids, one for each possible login security level.
    Each seid represents a particular secure path to the terminal.  Only
    one path to the terminal may do IO operations at a time, and this path
    is specified by the value of the V-function 'FCAcurrentPath(t)', where
    t refers to the terminal group or physical terminal.
    The different paths associated with a particular physical terminal
    are specified by the value of the V-function 'FCAterminalPathSet(t)',
    where t is as above.")

$(" EXTENTS -- Extents are addressable devices, representing areas on a
    disk, with a block size of 512 and a size determined when the device is
    physically mounted.  For the
    purposes of this specification, the size has been predetermined, as no
    physical mounting is specified.  The special property of extents is that
    they can be logically mounted, so that they `become' a file system
    or set of files.  When
    the system is started up, there are no files, except the root, only
    extents.  Each extent is mounted, setting up the actual file system
    that a user sees when he logs on.  When the extent is mounted, it can
    no longer be accessed as an extent.  Unmounting turns a file system
    back into an extent, and the file system disappears.")

$(" FILES -- Files are addressable devices, with a block size of 512 and
    two important properties.  They can be dynamically created and deleted,
    and they are of variable size.  Writing onto the end of a file effectively
    changes its size.  Files may also be linked to.  A link is a reference
    count used by the directory manager sitting above the kernel.  It
    represents the number of directories in which a file is found.  When
    this count goes to 0, and no process has the file open, the file
    is deleted.  This is the only way of deleting files, although they can
    be explicitly created.")

$(" SUBTYPES -- This is an additional protection mechanism over and above
    that provided by the mandatory, privilege, and discretionary access
    control systems.  Each openable object may be associated with a
    subtype, of which there are a fixed number at system generation time.
    Any object of a non-null subtype may be accessed only by those processes
    who have access rights to the subtype as well as the object.  The
    access right to a subtype is established by opening the subtype for
    the desired access.  Access to the subtype is granted or denied according
    to the usual mandatory and discretionary rules.  An object with
    a non-null subtype can be accessed only when the open descriptor for
    the subtype, to which access must already have been granted, is
    presented, and when the desired access to the object is a subset of
    the access granted to the subtype.  This forms a mini-capability
    mechanism with type extention, which is necessary for achieving access
    control over objects such as directories.  Thus there will be a
    directory subtype, to prevent arbitrary programs from damaging the
    directory system just because they have access to a particular
    directory.  The rules for subtype access occur in the open function and
    all functions that require a subtype capability for accessing
    an object.")


    TYPES

    $(FROM smx)
nonDisType: STRUCT_OF(
              INTEGER securityLevel; SET_OF securityCat securityCatS;
              INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
daType: SET_OF daMode;
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
tiiStruct: STRUCT_OF(nonDisType nd; modeStruct ms; INTEGER owner, group;
                    SET_OF privType priv);

    $(from fca ** exportable)
openDescriptor: DESIGNATOR;
openModes: {omRead, omWrite, omExclusiveRead, omExclusiveWrite};
IOfunction: {rewind, etc};     $(names for special kinds of IO functions)
deviceType: {RK05, RWP04, RWP05, RWP06, RSW04, TWE16, TM11, TU56, PR11,
           PC11, LP11, IMP11B, LHDH};
terminalGroup: DESIGNATOR;
fileNsp: {129 ..255}; $(nsp values for file systems)

    $(from fca ** redeclarable)
fileStatus: STRUCT_OF(INTEGER nBlocks, linkCount, timeLastMod;
                     seid subtype; BOOLEAN openAtCrash);
  $(data about an openable object that is returned to the user)
globalData: STRUCT_OF(INTEGER linkCount, openCount, timeLastMod; seid subtype;
                     BOOLEAN openAtCrash);
  $(state information for all openable objects)
fileBlock: VECTOR_OF CHAR;
ioStatus: STRUCT_OF(INTEGER devIndep, devDep);
  $(result of an IO operation, including possible hardware failure)
openFileEntry: STRUCT_OF(seid openSeid; SET_OF openModes openMode);
  $(entry in a process' open file table)
asyncId: CHAR;
readResult: STRUCT_OF(VECTOR_OF fileBlock data; ioStatus errst);
deviceStruct: STRUCT_OF(BOOLEAN addressable;
                       INTEGER minRequest, maxRequest, modSize, maxBlockNo);
  $(properties necessary for processing IO requests for devices)
mountTableEntry: STRUCT_OF(seid devSeid; BOOLEAN readOnly;
                          tiiStruct devTii; globalData devGl);
fileSystemEntry: STRUCT_OF(seid fileSeid; globalData gl; tiiStruct tii;
                          VECTOR_OF fileBlock fileData);
  $(the state of a file, for purposes of mounting and unmounting)
fileSystem: SET_OF fileSystemEntry;
  $(the state of an entire mountable file system)
sodPair: STRUCT_OF(seid ps; openDescriptor od);
  $(for openCount definition)


    PARAMETERS

SET_OF seid subtypeSeidSet; $( set of non-null subtypes known to the systsm)
seid nullStSeid; $( seid indicating the null subtype)
```

```
openDescriptor FCAnullSt; $(a file descriptor indicating the null subtype)
seid FCArootSeid; $(distinguished root of KSOS permanent file system)
INTEGER FCAmaxOpenDescriptors; $(maximum number of open descriptors per
                                  process)


    DEFINITIONS

INTEGER FCAfileSize(seid fSeid) IS
  CARDINALITY({INTEGER i | FCAfileData(fSeid, i) ~= ?});
  $(the size, in blocks, of an addressable device, file, or extent)

INTEGER nOpenDescriptors(seid pSeid) IS
  CARDINALITY({openDescriptor od | FCAopenEntry(pSeid, od) ~= ?});
  $(the number of open objects in a given process; it must not exceed a fixed
    maximum)

deviceStruct deviceDataSeid(seid fSeid) IS
  DDFdeviceData(FCAdeviceType(fSeid));
  $(given the seid of a device, the data on which its IO requests depend)

SET_OF daMode h_modeTrans(SET_OF openModes oModes) IS
  (IF omRead INSET oModes THEN {daRead} ELSE {})
    UNION (IF omWrite INSET oModes THEN {daWrite} ELSE {});
  $(translates from one enumerated type, open Modes, to another slightly
    different enumerated type, discretionary access modes)

BOOLEAN isCurrentPath(seid tSeid) IS
  EXISTS terminalGroup t : FCAcurrentPath(t) = tSeid;
  $(for a given terminal, tells whether it is the active path to its physical
    device)

BOOLEAN isReadOnly(seid s) IS
  EXISTS fileNsp fsNsp
    : SENseidNsp(FCAmountTable(fsNsp).devSeid) = SENseidNsp(s)
        AND FCAmountTable(fsNsp).readOnly = TRUE;
  $(tells whether or not a given file is on a file system that is mounted in
    read-only mode)

$(When a file's link count goes to zero, its link count becomes '?', which
  means that it does not exist for any operation, including K_link.  The
  file cannot be physically deleted until its open count becomes zero, though.)

   BOOLEAN logicalFileExistence(seid fSeid) IS
        FCAinfo(fSeid) ~= ? AND FCAinfo(fSeid).linkCount ~= ?;

    EXTERNALREFS

    FROM mac:
VFUN MACclock() -> INTEGER time;

    FROM smx:
seid: DESIGNATOR;
secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
                   tExtent, tNull};
        privType: {
```

```
                          privFileUpdateStatus,  privLink,
                          privLockSeg,           privModifyPriv,
                          privMount,             privSetLevel,
                          privStickySeg,         privSetPath,
                          privViolSimpSecurity,  privViolStarSecurity,
                          privViolSimpIntegrity, privViolStarIntegrity,
                          privViolDiscrAccess,   privRealizeExecPermission,
                          privSignal,            privWalkPTable,
                          privHalt,              privKernelCall,
                          privViolCompartments,  privSetComm,
                          privImmigrate
                          };

daMode: {daRead, daWrite, daExecute};
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;
VFUN SENseidNsp(seid s) -> INTEGER nsp;
VFUN SENseidType(seid s) -> secureEntityType set;
VFUN TIIinfo(seid s) -> tiiStruct tiist;
VFUN SMXhasPriv(seid pSeid; privType priv) -> BOOLEAN b;
VFUN SMXflow(seid pSeid, oSeid; daType d) -> BOOLEAN b;
VFUN SMXdap(seid pSeid, oSeid; daType d) -> BOOLEAN b;

    FROM ddf:
VFUN DDFdeviceData(deviceType d) -> deviceStruct ds;         $(DDFdeviceData)



    ASSERTIONS

FORALL seid s; openDescriptor od
  : SMXflow(s, FCAopenEntry(s, od).openSeid, {daRead});
  $(FCAopenEntry only contains seids of objects whose security
    levels are lower than that of the opening process)

FORALL seid pSeid;
       openDescriptor od | FCAopenEntry(pSeid, od) ~= ?;
       openFileEntry oe = FCAopenEntry(pSeid, od)
  : (omWrite INSET oe.openMode
         AND SENseidType(oe.openSeid) ~= tSubtype)
      => SMXflow(pSeid, oe.openSeid, {daRead, daWrite});
   $( any object open for reading and writing that is not a subtype
        must be at the same security level as the process that opened
        it)
FORALL seid pSeid;
       openDescriptor od | FCAopenEntry(pSeid, od) ~= ?;
       openFileEntry oe = FCAopenEntry(pSeid, od)
  : {omExclusiveRead, omExclusiveWrite} SUBSET oe.openMode
      => SMXflow(pSeid, oe.openSeid, {daRead, daWrite});
   $(all files open for exclusive use are open only by processes of
       the same security level)

FORALL seid s | FCAinfo(s) ~= ? AND FCAinfo(s).subtype ~= nullStSeid
  : SMXflow(s, FCAinfo(s).subtype, {daRead});
   $( the level of a subtype is less than or equal to the levels
     of any objects of that subtype)
```

```

FORALL seid fSeid | FCAinfo(fSeid) ~= ?
  : SENseidType(fSeid) INSET {tTerminal, tDevice, tSubtype, tFile, tExtent};
  $(restricts the types of objects manipulated by this module)

FORALL seid fSeid: {INTEGER i | FCAfileData(fSeid, i) ~= ?}
                       = {0 .. FCAfileSize(fSeid) - 1};
  $(the blocks of a file, extent, or addressable device form a sequence)

FORALL seid dSeid
  | SENseidType(dSeid) = tDevice AND FCAinfo(dSeid) ~= ?
  : (LET deviceStruct d = deviceDataSeid(dSeid)
       IN NOT d.addressable => d.modSize = 1 AND d.maxBlockNo = 0);
  $(necessary properties of all non-addressable devices)

FORALL seid fSeid
  | FCAinfo(fSeid) ~= ?
  : (LET secureEntityType t = SENseidType(fSeid)
       IN FORALL INTEGER i | FCAfileData(fSeid, i) ~= ?
           : LENGTH(FCAfileData(fSeid, i))
               = (IF t = tDevice THEN deviceDataSeid(fSeid).modSize
                  ELSE IF t INSET {tExtent, tFile} THEN 512
                     ELSE ?));
  $(the lengths of all blocks for files, extents, or addressable devices
    must correspond to the paremeters for those objects)

FORALL seid s | SENseidType(s) = tTerminal AND FCAinfo(s) ~= ?
  : (EXISTS terminalGroup t1
       : s INSET FCAterminalPathSet(t1)
           AND (FORALL terminalGroup t2 ~= t1
                  : NOT s INSET FCAterminalPathSet(t2)));
  $(each terminal is in exactly one terminal group)

FORALL seid f
     | EXISTS INTEGER i : FCAfileData(f, i) ~= ?
  : FCAinfo(f) ~= ?
      AND (SENseidType(f) = tDevice AND deviceDataSeid(f).addressable = TRUE
             OR SENseidType(f) INSET {tFile, tExtent});
  $(only files, extents, or addressable devices have file data)

FORALL seid f
  : (FCAinputStream(f) ~= ? AND FCAoutputStream(f) ~= ?)
       = (SENseidType(f) = tTerminal
            OR SENseidType(f) = tDevice
                 AND deviceDataSeid(f).addressable = FALSE);
   $(non-addressable device have both an input and an output stream, although
     it may always be null)

FORALL seid pSeid | FCAopenTableExists(pSeid)
  : FCAopenEntry(pSeid, FCAnullSt) = ?;
  $(there are never any opened objects assigned to the open descriptor
    reserved for the null subtype)

FORALL seid s1; seid s2
  : SENseidType(s1) = tSubtype AND SENseidType(s2) = tSubtype
       => SENseidNsp(s1) = SENseidNsp(s2);
  $(subtypes have only one name space partition)

SENseidType(nullStSeid) = tSubtype;
  $(the null subtype seid is a subtype )

FORALL seid s INSET subtypeSeidSet : SENseidType(s) = tSubtype;
  $(properties of the set of non*null subtypes)

SENseidType(FCArootSeid) = tFile;
  $(property of the distinguished root of the entire file system)

FORALL seid f | FCAinfo(f) ~= ? AND SENseidType(f) = tFile
  :EXISTS fileNsp fsNsp :SENseidNsp(FCAmountTable(fsNsp).devSeid)
                        = SENseidNsp(f)
                         OR f = FCArootSeid;
  $(a file is either the root or it is on a mountable file system)


    FUNCTIONS

$(********************* state functions ** devices *************************)

VFUN FCAdeviceType(seid fSeid) *> deviceType d;              $(FCAdeviceType)
  $(gives the type of a particular device, which determines its IO behavior)
  HIDDEN;
  INITIALLY
    (d ~= ?)
      = (SENseidType(fSeid) = tDevice AND FCAinfo(fSeid) ~= ?);

$(*********************** state functions ** terminals ******************)

VFUN FCAterminalPathSet(terminalGroup t) *> SET_OF seid ss;
                                                     $(FCAterminalPathSet)
  $(defines the set of paths, or logical terminals, that correspond to
    a given terminal group, or physical terminal)
  HIDDEN;
  INITIALLY
    FORALL seid s INSET ss
      : FCAinfo(s) ~= ? AND SENseidType(s) = tTerminal;

VFUN FCAcurrentPath(terminalGroup t) *> seid s;            $(FCAcurrentPath)
  $(the seid of the logical terminal that is allowed to do IO on a given
    physical terminal)
  HIDDEN;
  INITIALLY
    s INSET FCAterminalPathSet(t);

$(****************** state functions ** mountable file systems ***********)

VFUN FCAmountTable(fileNsp fsNsp) *> mountTableEntry mte;  $(FCAmountTable)
  $(for a given extent that is mounted, tells leaf of the old file system,
    the root of the new or mounted file system, whether the file system
    is read only, and the state information for the extent)
  HIDDEN;
  INITIALLY mte = ?;

```

```
VFUN FCAextentToFileSys(VECTOR_OF fileBlock fb; seid rootSeid)
  *> fileSystem fs;                                  $(FCAextentToFileSys)
  $(given the data of a given extent and a root seid for a file system
    to be made out of the extent whose data is given, produces the
    file system, consisting of a set of tuples each made up of a
    file seid and the state of the file)
  HIDDEN;
  INITIALLY
    fs ~= ? =>
      (EXISTS fileSystemEntry fse INSET fs : rootSeid = fse.fileSeid)
        AND (FORALL fileSystemEntry fse INSET fs
               : SENseidNsp(fse.fileSeid) = SENseidNsp(rootSeid));
    $(a constant function for data conversion)

$(******************* state functions ** files **************)

VFUN FCAinfo(seid fSeid) *> globalData gl;                   $(FCAinfo)
  $(the status information, excluding data and type dependent stuff, for
    an openable object)
  HIDDEN;
  INITIALLY
    IF deviceDataSeid(fSeid) ~= ? THEN gl ~= ?
    ELSE IF fSeid INSET {FCArootSeid, nullStSeid}
      THEN gl.subtype = nullStSeid
    ELSE IF fSeid INSET subtypeSeidSet THEN gl.subtype = nullStSeid
    ELSE gl = ?;

VFUN FCAfileData(seid fSeid; INTEGER blockNo) *> fileBlock fb; $(FCAfileData)
  $(the data contained on a file, extent or an addressable device)
  DEFINITIONS
    secureEntityType type IS SENseidType(fSeid);
    deviceStruct d
      IS IF type = tDevice THEN deviceDataSeid(fSeid)
           ELSE IF type INSET {tFile, tExtent}
             THEN STRUCT(TRUE, 512, 65534, 512, FCAfileSize(fSeid)*1)
           ELSE STRUCT(FALSE, 1, 255, 1, 0);
  HIDDEN;
  INITIALLY
    (IF FCAinfo(fSeid) = ? OR type INSET {tTerminal, tSubtype}
           OR NOT blockNo INSET {0 .. d.maxBlockNo} OR NOT d.addressable
         THEN fb = ?
         ELSE fb ~= ?)
       AND (fb ~= ?
              => LENGTH(fb) = d.modSize
                   AND (FORALL INTEGER i INSET {0 .. blockNo * 1}
                          : FCAfileData(fSeid, i) ~= ?)
                   AND (FORALL INTEGER j INSET {1 .. LENGTH(fb)}
                          : fb[j] ~= ?));

VFUN FCAinputStream(seid fSeid) *> VECTOR_OF CHAR vc;      $(FCAinputStream)
  $(the input data for terminals and nonaddressable devices)
  HIDDEN;
  INITIALLY
    IF FCAinfo(fSeid) ~= ?
         AND (SENseidType(fSeid) = tTerminal
                OR SENseidType(fSeid) = tDevice
```

```
                          AND deviceDataSeid(fSeid).addressable = TRUE)
             THEN vc ~= ?
             ELSE vc = ?;

VFUN FCAoutputStream(seid fSeid) *> VECTOR_OF CHAR vc;      $(FCAoutputStream)
  $(the output data for terminals and nonaddressable devices)
  HIDDEN;
  INITIALLY
    IF FCAinfo(fSeid) ~= ?
         AND (SENseidType(fSeid) = tTerminal
                   OR SENseidType(fSeid) = tDevice
                          AND deviceDataSeid(fSeid).addressable = TRUE)
       THEN vc ~= ?
       ELSE vc = ?;

$(*********************** state functions ** open tables *****************)

VFUN FCAopenTableExists(seid pSeid) *> BOOLEAN b;       $(FCAopenTableExists)
  $(the existence predicate for the table of openable objects corresponding
    to a given process)
  HIDDEN;
  INITIALLY b = FALSE;

VFUN FCAopenEntry(seid pSeid; openDescriptor od) *> openFileEntry oe;
  $(the information in entry "od" of the open table of process "pSeid")
  HIDDEN;                                                       $(FCAopenEntry)
  INITIALLY oe = ?;

$(******************** creation of files ***********************************)

OVFUN FCAcreate(seid pSeid, nspSeid; SET_OF openModes mode; modeStruct d; openDescriptor s
  *> STRUCT_OF(seid fSeid; openDescriptor od) return;            $(FCAcreate)

  $(creates a file and opens it in the desired mode.  the file system on
    which the file resides must be writable. If a subtype capability
    is provided, it must refer to a valid subtype.  The created file gets
    only the discretionary access permission specified.  All other data
    comes from the process making the call.  The file is originally of zero
    length)
  DEFINITIONS
    tiiStruct ptii IS TIIinfo(pSeid);
    tiiStruct otii IS STRUCT(ptii.nd, d, ptii.owner, ptii.group, ptii.priv);
    seid extSeid
      IS SOME seid s | SENseidNsp(FCAmountTable(SENseidNsp(s)).devSeid)
                           = SENseidNsp(nspSeid);
  EXCEPTIONS
    $(these exceptions subsume those for opening a file for writing)
    XbadModes: NOT ((mode = {omRead})
                   OR  (mode = {omWrite})
                   OR  (mode = {omRead, omWrite})
                   OR  (mode = {omRead, omWrite, omExclusiveWrite})
                   OR  (mode = {omRead, omWrite, omExclusiveRead, omExclusiveWrite}));
    XbadStCap:
      stCap ~= FCAnullSt AND SENseidType(FCAopenEntry(pSeid, stCap).openSeid) ~= tSubtype;

    XodSpace: nOpenDescriptors(pSeid) >= FCAmaxOpenDescriptors;

    XnoFile:                           $( actually means no file system )
      NOT(extSeid ~= ? AND SMXflow(pSeid, extSeid, {daRead}));
      $(the flow violation applies to the file system on which the newly
         created file is to reside; its TII exists but not its global data)
    XnotWritable: isReadOnly(nspSeid);
    XbadSubtype:                 $( NOT currently checked in implementation JN)
      stCap ~= FCAnullSt
        AND NOT omWrite INSET FCAopenEntry(pSeid, stCap).openMode;
    RESOURCE_ERROR;
  ASSERTIONS
    FCAopenTableExists(pSeid);
    SMXflow(pSeid, nspSeid, {daRead});
  EFFECTS
    LET seid fs | FCAinfo(fs) = ?;
        openDescriptor o | FCAopenEntry(pSeid, o) = ?
        AND o ~= FCAnullSt
        AND SENseidType(fs) = tFile
        AND SENseidNsp (fs) = SENseidNsp(nspSeid)
        IN 'TIIinfo(fs) = otii
        AND 'FCAopenEntry(pSeid, o) = STRUCT(fs, mode)
        AND return = STRUCT(fs,o)
        AND 'FCAinfo(fs) = STRUCT(0, 1, MACclock(),
                                 IF stCap = FCAnullSt THEN nullStSeid
                                   ELSE FCAopenEntry(pSeid, stCap).openSeid,
                                 FALSE);

$(*********************** file opening and closing ***********************)

OVFUN FCAopen(seid pSeid, o; SET_OF openModes mode; openDescriptor stCap)
        *> openDescriptor od;                                     $(FCAopen)
  $(opens the openable object specified by "o".  The object must exist
    and the process "pSeid" must have the right mandatory and discretionary
    access.  Special rules, described below, apply to subtypes.
    Special rules also apply when the object is being opened for exclusive
    use.  Only one process may open an object for exclusive use.
    A process is allowed a fixed maximum number of open objects.)
  DEFINITIONS
    globalData ofst IS FCAinfo(o);
    openFileEntry stEntry IS FCAopenEntry(pSeid, stCap);

  EXCEPTIONS
    XbadModes: NOT ((mode = {omRead})          $( bad user argument )
                OR  (mode = {omWrite})
                OR  (mode = {omRead, omWrite})
                OR  (mode = {omRead, omWrite, omExclusiveWrite})
                OR  (mode = {omRead, omWrite, omExclusiveRead, omExclusiveWrite}));

    XbadStCap:                                 $( bad subtype od given )
      stCap ~= FCAnullSt AND SENseidType(stEntry.openSeid) ~= tSubtype;
      $(a subtype capability was specified, but it is not for a valid subtype)

    XodSpace: nOpenDescriptors(pSeid) >= FCAmaxOpenDescriptors;
                                               $( no open descriptors left )

    XnoFile: (ofst = ?)                        $( inaccessable )
```

```
          OR (omRead  INSET mode AND NOT SMXflow(pSeid, o, {daRead }))
          OR (omWrite INSET mode AND NOT SMXflow(pSeid, o, {daWrite}))
          OR (SENseidType(o) = tFile AND NOT logicalFileExistence(o));
                                                $( discretionary access ck )
    XdapViol: NOT SMXdap(pSeid, o, h_modeTrans(mode));

    XbadSubtype1:
      stCap = FCAnullSt AND NOT ofst.subtype = nullStSeid;
      $(no subtype capability was specified, but the object has a non-null
        subtype)

    XbadSubtype2:                                $( unwanted subtype seid )
      stCap ~= FCAnullSt AND ofst.subtype = nullStSeid;

    XbadSubtype3:
      stCap ~= FCAnullSt
        AND (stEntry.openSeid ~= ofst.subtype   $( wrong subtype )
               OR NOT (         $( open modes must be subset of subtype modes)
                       (mode INTER {omRead, omWrite})
                       SUBSET (stEntry.openMode INTER {omRead, omWrite})
                      )
            );
       $(a subtype capability is specified, but it does not match the subtype
         of the object, with access modes included;
         note that execute mode on the subtype is required to write on
         an object of the subtype)
    XnotWritable: omWrite INSET mode AND isReadOnly(o);
    Xbusy:                                  $( exclusive use conflict )
      EXISTS seid pSeid1; openDescriptor od1.
        FCAopenEntry(pSeid1, od1).openSeid = o AND
          (    (omRead INSET mode
                AND omExclusiveRead INSET FCAopenEntry(pSeid1,od1).openMode)
            OR (omWrite INSET mode
                AND omExclusiveWrite INSET FCAopenEntry(pSeid1,od1).openMode)
            OR (omExclusiveRead INSET mode
                AND omRead INSET FCAopenEntry(pSeid1,od1).openMode)
            OR (omExclusiveWrite INSET mode
                AND omWrite INSET FCAopenEntry(pSeid1,od1).openMode)
          );
  ASSERTIONS
    FCAopenTableExists(pSeid);
  EFFECTS
    LET openDescriptor od1 | FCAopenEntry(pSeid, od1) = ? AND od1 ~= FCAnullSt
      IN 'FCAopenEntry(pSeid, od1) = STRUCT(o, mode)
      AND od = od1;
    'FCAinfo(o) = STRUCT(ofst.linkCount, ofst.openCount + 1,
                             ofst.timeLastMod, ofst.subtype, ofst.openAtCrash);

OFUN FCAclose(seid pSeid; openDescriptor od);                     $(FCAclose)
  $(closes the open object named by "od".  If the object is not is use by
    anyone else, either by linking or by opening, then the object is deleted)
  DEFINITIONS
    openFileEntry oe IS FCAopenEntry(pSeid, od);
    seid fSeid IS oe.openSeid;
    globalData ofst IS FCAinfo(fSeid);
  EXCEPTIONS
```

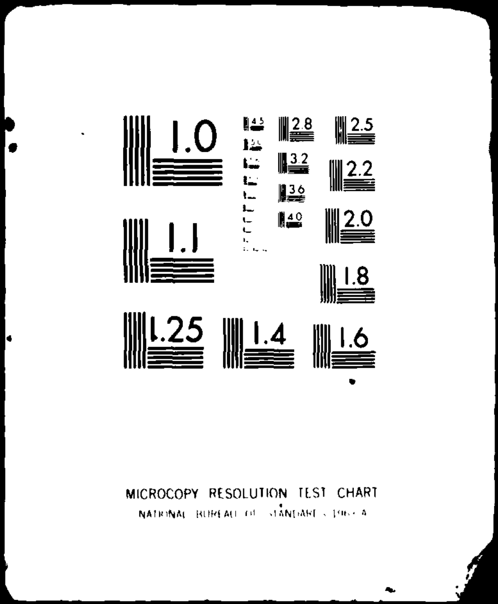
```
    XbadOd: oe = ?;
  ASSERTIONS
    FCAopenTableExists(pSeid);
  EFFECTS
    IF (ofst.linkCount = 0 OR ofst.linkCount = ?)
        AND ofst.openCount = 1
        AND SENseidType(fSeid) = tFile
      THEN 'FCAinfo(fSeid) = ?
            AND (FORALL INTEGER i : 'FCAfileData(fSeid, i) = ?)
            AND 'TIIinfo(fSeid) = ?
      ELSE 'FCAinfo(fSeid) = STRUCT(ofst.linkCount, ofst.openCount + 1,
                                    ofst.timeLastMod, ofst.subtype,
                                    ofst.openAtCrash);
    'FCAopenEntry(pSeid, od) = ?;
    $(the openAtCrash field is cleared when closing an object that was open
      for writing.  This has not been put in.)

$(*************************** open table maintenance ***********************)

OFUN FCAcreateOpenTable(seid pSeid);                    $(FCAcreateOpenTable)
  $( this operation creates an open table associated with a process seid;
     it is used as an auxiliary operation by the process modules when
     creating a new a process)
  ASSERTIONS
    NOT FCAopenTableExists(pSeid);
  EFFECTS
    'FCAopenTableExists(pSeid) = TRUE;

OFUN FCAdeleteOpenTable(seid pSeid);                    $(FCAdeleteOpenTable)
  $( Deletes the open table associated with a process; supports the release
     of a process)
  ASSERTIONS
    FCAopenTableExists(pSeid);
  EFFECTS
    'FCAopenTableExists(pSeid) = FALSE;

$(************************ utility operations ******************************)

OFUN FCAcopyOpenTable(seid fromSeid, toSeid);            $(FCAcopyOpenTable)
  $(Copies the contents of one open table, "fromSeid," to another, "toSeid."
    "toSeid" must be empty)
  ASSERTIONS
    FCAopenTableExists(fromSeid);
    FCAopenTableExists(toSeid);
    FORALL openDescriptor od : FCAopenEntry(toSeid, od) = ?;
  EFFECTS
    FORALL openDescriptor od;
           seid fSeid = FCAopenEntry(fromSeid, od).openSeid;
           globalData ofst = FCAinfo(fSeid)
      : 'FCAopenEntry(toSeid, od) = FCAopenEntry(fromSeid, od)
          AND (ofst ~= ?
                 => 'FCAinfo(fSeid)
                      = STRUCT(ofst.linkCount, ofst.openCount + 1,
                               ofst.timeLastMod, ofst.subtype,
                               ofst.openAtCrash));

```

```
$(*********************** file status operations **************************)

VFUN FCAgetFileStatus(seid pSeid, fSeid) -> fileStatus fst;$(FCAgetFileStatus)
  $(returns the status of the file.  The requesting process must have
    mandatory access to the object, and the object must exist)
  DEFINITIONS
    seid f IS fSeid;
    globalData gl IS FCAinfo(f);
  EXCEPTIONS
    XnoFile: NOT(FCAinfo(f) ~= ? AND SMXflow(pSeid, f, {daRead}));
  DERIVATION
    STRUCT(IF FCAfileSize(f) = ? THEN 0 ELSE FCAfileSize(f),
           gl.linkCount, gl.timeLastMod, gl.subtype,
           gl.openAtCrash);

END_MODULE
```

SSIFIE

2 OF 4

AD
A111578


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

```
pbl.specs     Page 2     Wed Mar 18 11:12:29 1981

 57       FROM pvm:
```

```
$("     MODULE:         fmi.specs (version 4.14)
        CONTENTS:       File Miscellaneous Operations
        TYPE:           SPECIAL specifications
        LAST CHANGED:   12/23/80, 16:48:40
  ")


MODULE fmi


$( this module contains miscellaneous file operations that are not included
   in the fca module, because the SPECIAL checker at FACC cannot accomodate
   the combined file)

    TYPES

    $(FROM smx)
nonDisType: STRUCT_OF(
               INTEGER securityLevel; SET_OF securityCat securityCatS;
               INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
daType: SET_OF daMode;
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
tiiStruct: STRUCT_OF(nonDisType nd; modeStruct ms; INTEGER owner, group;
                     SET_OF privType priv);

    $(from fca)
globalData: STRUCT_OF(INTEGER linkCount, openCount, timeLastMod;
                      seid subtype; BOOLEAN openAtCrash);
ioStatus: STRUCT_OF(INTEGER devIndep, devDep);
asyncId: CHAR;
fileBlock: VECTOR_OF CHAR;
openFileEntry: STRUCT_OF(seid openSeid; SET_OF openModes openMode);
readResult: STRUCT_OF(VECTOR_OF fileBlock data; ioStatus errst);
deviceStruct: STRUCT_OF(BOOLEAN addressable;
                        INTEGER minRequest, maxRequest, modSize, maxBlockNo);
mountTableEntry: STRUCT_OF(seid devSeid; BOOLEAN readOnly;
                           tiiStruct devTii; globalData devGl);
fileSystemEntry: STRUCT_OF(seid fileSeid; globalData gl; tiiStruct tii;
                           VECTOR_OF fileBlock fileData);
fileSystem: SET_OF fileSystemEntry;

fileNsp: {129 .. 255};                $( nsp values for file systems )

    DEFINITIONS

$(these definitions are explained in the fca module)

INTEGER FCAfileSize(seid fSeid) IS
  CARDINALITY({INTEGER i | FCAfileData(fSeid, i) ~= ?});

deviceStruct deviceDataSeid(seid fSeid) IS
  DDFdeviceData(FCAdeviceType(fSeid));

BOOLEAN isCurrentPath(seid tSeid) IS
  EXISTS terminalGroup t : FCAcurrentPath(t) = tSeid;

```

```
BOOLEAN isReadOnly(seid fSeid) IS
        FCAmountTable(SENseidNsp(fSeid)).readOnly = TRUE;

$(When a file's link count goes to zero, its link count becomes '?', which
  means that it does not exist for any operation, including K_link.  The
  file cannot be physically deleted until its open count becomes zero, though.)

   BOOLEAN logicalFileExistence(seid fSeid) IS
        FCAinfo(fSeid) ~= ? AND FCAinfo(fSeid).linkCount ~= ?;

    EXTERNALREFS

    FROM ddf:

VFUN DDFdeviceData(deviceType d) -> deviceStruct ds;

    FROM smx:
seid: DESIGNATOR;
secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
                   tExtent, tNull};
        privType: {
                          privFileUpdateStatus,  privLink,
                          privLockSeg,           privModifyPriv,
                          privMount,             privSetLevel,
                          privStickySeg,         privSetPath,
                          privViolSimpSecurity,  privViolStarSecurity,
                          privViolSimpIntegrity, privViolStarIntegrity,
                          privViolDiscrAccess,   privRealizeExecPermission,
                          privSignal,            privWalkPTable,
                          privHalt,              privKernelCall,
                          privViolCompartments,  privSetComm,
                          privImmigrate
                          };
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;
daMode: {daRead, daWrite, daExecute};
VFUN SENseidNsp(seid s) -> INTEGER nsp;
VFUN SENseidType(seid s) -> secureEntityType set;
VFUN TIIinfo(seid s) -> tiiStruct tiist;
VFUN SMXhasPriv(seid pSeid; privType priv) -> BOOLEAN b;
VFUN SMXflow(seid pSeid, oSeid; daType d) -> BOOLEAN b;
VFUN SMXdap(seid pSeid, oSeid; daType d) -> BOOLEAN b;

    FROM fca:
openDescriptor: DESIGNATOR;
openModes: {omRead, omWrite, omExclusiveRead, omExclusiveWrite};
IOfunction: {rewind, etc};
deviceType: {RK05, RWP04, RWP05, RWP06, RSW04, TWE16, TM11, TU56, PR11,
             PC11, LP11, IMP11B, LHDH};
terminalGroup: DESIGNATOR;
seid FCArootSeid;
VFUN FCAdeviceType(seid fSeid) -> deviceType d;
VFUN FCAcurrentPath(terminalGroup t) -> seid s;
VFUN FCAterminalPathSet(terminalGroup t) -> SET_OF seid ss;
VFUN FCAmountTable(fileNsp nsp) -> mountTableEntry mte;
VFUN FCAextentToFileSys(VECTOR_OF fileBlock fb; fileNsp nsp)
```

```
  *> fileSystem fs;
VFUN FCAinfo(seid fSeid) *> globalData gl;
VFUN FCAfileData(seid fSeid; INTEGER blockNo) *> fileBlock fb;
VFUN FCAinputStream(seid fSeid) *> VECTOR_OF CHAR vc;
VFUN FCAoutputStream(seid fSeid) *> VECTOR_OF CHAR vc;
VFUN FCAopenTableExists(seid pSeid) *> BOOLEAN b;
VFUN FCAopenEntry(seid pSeid; openDescriptor od) *> openFileEntry oe;


    FUNCTIONS

$(*********************** process support operations *********************)

$(Note: the function FCAcreateOpenTable is sufficient to support PROspawn.
  The other support operations are listed here.)

OFUN FMIforkSupport(seid parent, child);                   $(FMIforkSupport)
  $(This function creates a new open table, "child," and copies all open
    from "parent" into it)
  EXCEPTIONS
    XexclusiveFile:
      EXISTS openDescriptor od
        : {omExclusiveRead,omExclusiveWrite} SUBSET
                 FCAopenEntry(parent, od).openMode;
  ASSERTIONS
    FCAopenTableExists(parent);
    NOT FCAopenTableExists(child);
  EFFECTS
    'FCAopenTableExists(child) = TRUE;
    FORALL openDescriptor od;
           openFileEntry oe = FCAopenEntry(parent, od);
           globalData gl = FCAinfo(oe.openSeid)
      : 'FCAopenEntry(child, od) = oe
          AND 'FCAinfo(oe.openSeid)
                = STRUCT(gl.linkCount, gl.openCount + 1, gl.timeLastMod,
                         gl.subtype, gl.openAtCrash);


OFUN FMIreleaseSupport(seid pSeid);                      $(FCAreleaseSupport)
  $( Closes all the open objects of an open object table and deletes the
     table)
  DEFINITIONS
    INTEGER nOpen(openDescriptor od)
      IS CARDINALITY({openDescriptor od1
                        | FCAopenEntry(pSeid, od).openSeid
                             = FCAopenEntry(pSeid, od1).openSeid});
      $(the number of times an object corresponding to a
          given open descriptor is opened by this process)
    seid s(openDescriptor od) IS FCAopenEntry(pSeid, od).openSeid;
  ASSERTIONS
    FCAopenTableExists(pSeid);
  EFFECTS
    FORALL openDescriptor od | FCAopenEntry(pSeid, od) ~= ?;
           globalData gl = FCAinfo(s(od))
      : 'FCAopenEntry(pSeid, od) = ?
          AND (IF( gl.linkCount = 0  OR gl.linkCount =?)
                   AND gl.openCount = nOpen(od)
                   AND SENseidType(s(od)) = tFile
                      THEN 'FCAinfo(s(od)) = ?
                        AND 'TIIinfo(s(od)) = ?
                        AND (FORALL INTEGER i : 'FCAfileData(s(od), i) = ?)
                      ELSE
                        'FCAinfo(s(od))
                          = STRUCT(gl.linkCount,
                                   gl.openCount - nOpen(od),
                                   gl.timeLastMod, gl.subtype,
                                   gl.openAtCrash));
    'FCAopenTableExists(pSeid) = FALSE;

$(*********************** link and unlink operations *********************)

OFUN FCAlink(seid pSeid, f);                                        $(FCAlink)
  $(increments the link count of an existing file.  Mandatory access is
    required, and the process must have the privilege to link.  The file
    system must not be mounted in read only mode)
  DEFINITIONS
    globalData fs IS FCAinfo(f);
  EXCEPTIONS
    XbadPriv: NOT SMXhasPriv(pSeid, privLink);
    XnoFile: NOT(logicalFileExistence(f) AND SMXflow(pSeid, f, {daWrite}));
    XnoFile: SENseidType(f) ~= tFile;   $( not a file, but exists )
    XnotWritable: isReadOnly(f);
  ASSERTIONS
    FCAopenTableExists(pSeid);
  EFFECTS
    'FCAinfo(f)
      = STRUCT(fs.linkCount + 1, fs.openCount, fs.timeLastMod,
               fs.subtype, fs.openAtCrash);

OFUN FCAunlink(seid pSeid, f);                                    $(FCAunlink)
 $(decrements the reference count of an existing file. Mandatory access is
   required and the process must have the privilege to link.  The file
   system that the file is on must not be mounted in read only mode.  Note
   that unlinking can cause the file to be deleted if the link count goes
   to zero and the file is not open anywhere)
  DEFINITIONS
    globalData fs IS FCAinfo(f);
  EXCEPTIONS
    XbadPriv: NOT SMXhasPriv(pSeid, privLink);
    XnoFile: NOT(logicalFileExistence(f) AND SMXflow(pSeid, f, {daWrite}));
    XnoFile: SENseidType(f) ~= tFile;
    XnotWritable: isReadOnly(f);
  ASSERTIONS
    FCAopenTableExists(pSeid);
  EFFECTS
    IF ((fs.linkCount <= 1) OR (fs.linkCount = ?))
    AND fs.openCount = 0
                                          $(if true, then delete file)
      THEN 'FCAinfo(f) = ?
              AND (FORALL INTEGER i : 'FCAfileData(f, i) = ?)
              AND 'TIIinfo(f) = ?
      ELSE 'FCAinfo(f) =
        STRUCT((IF fs.linkCount <= 1 THEN ? ELSE fs.linkCount * 1),
          fs.openCount, fs.timeLastMod, fs.subtype, fs.openAtCrash);
$(*********************** basic I/O operations ****************************)

VFUN FCAvReadBlocks(seid pSeid; openDescriptor od; INTEGER blockNo, size;
          asyncId id)                         $(FCAvReadBlocks)
    *> readResult rr;
 $(the purpose of this function is to return the result that FCAreadBlocks
  would return if executed)
 $(returns blocks that are read from a given file, device, extent, or
  terminal.  the object must be open for reading. the block number and
  size must be within range for the kind of object specified.  note
  that files, extents, and addressable devices are handled differently
  from terminals and nonaddressable devices, with respect to the data.)
 DEFINITIONS
  seid fSeid IS FCAopenEntry(pSeid, od).openSeid;
  deviceStruct d
   IS IF SENseidType(fSeid) = tDevice THEN deviceDataSeid(fSeid)
        ELSE IF SENseidType(fSeid) INSET {tFile, tExtent}
         THEN STRUCT(TRUE, 512, 32768, 512, FCAfileSize(fSeid)*1)
        ELSE STRUCT(FALSE, 1, 255, 1, 0); $(tTerminal)
  INTEGER blockCount IS size/512;
  INTEGER blockPast IS (IF SENseidType(fSeid) = tDevice THEN
                          MIN({d.maxBlockNo, blockNo+blockCount})
                         ELSE
                         MIN({FCAfileSize(fSeid),blockNo+blockCount}));

 EXCEPTIONS
  XbadOd: FCAopenEntry(pSeid, od) = ?;
  XbadSize: size = 0;
  XnotReadable: NOT omRead INSET FCAopenEntry(pSeid, od).openMode;
  $( EXCEPTIONS AFTER THIS POINT ARE ACTUALLY HANDLED AS I/O ERRORS;
     In addition, the following exceptions may not be treated as "pure"
     in the implementation, i.e., they may involve effects in the code.)
  XbadBlockNo:
   NOT size INSET {d.minRequest .. d.maxRequest}
    OR (size MOD d.modSize) ~= 0;
  XbadBlockNo: NOT blockNo INSET {0 .. d.maxBlockNo};
  XendOfFile:
   d.addressable AND blockNo + size/d.modSize > d.maxBlockNo;
  Xerror: RESOURCE_ERROR;
 ASSERTIONS
  FCAopenTableExists(pSeid);
 DERIVATION
  IF d.addressable THEN
     STRUCT(
           VECTOR(FOR i FROM blockNo TO blockPast * 1
                  : FCAfileData(fSeid, i)),
           SOME ioStatus ios | TRUE)
   ELSE
    LET INTEGER sl INSET {0 .. size/d.modSize}
     $(Must let sl be nondeterministically selected for nonaddress. dev)
    IN STRUCT(
           VECTOR(FOR i FROM 1 TO sl $(d.modSize = 1)
                  : VECTOR(FCAinputStream(fSeid)[i])),
          SOME ioStatus ios | TRUE);
```

```

OVFUN FCAreadBlocks(seid pSeid; openDescriptor od; INTEGER blockNo, size;
                    asyncId id)
        *> readResult rr;                                  $(FCAreadBlocks)
  $(LR ** needs semantics for asynchronous I/O)
  $(returns blocks that are read from a given file, device, extent, or
    terminal.  the object must be open for reading. the block number and
    size must be within range for the kind of object specified.  note
    that files, extents, and addressable devices are handled differently
    from terminals and nonaddressable devices, with respect to the data.)
  DEFINITIONS
    seid fSeid IS FCAopenEntry(pSeid, od).openSeid;
    deviceStruct d
      IS IF SENseidType(fSeid) = tDevice THEN deviceDataSeid(fSeid)
           ELSE IF SENseidType(fSeid) INSET {tFile, tExtent}
             THEN STRUCT(TRUE, 512, 32768, 512, FCAfileSize(fSeid)*1)
           ELSE STRUCT(FALSE, 1, 255, 1, 0); $(tTerminal)
    readResult rr1 IS FCAvReadBlocks(pSeid, od, blockNo, size, id);
  EXCEPTIONS
    EXCEPTIONS_OF FCAvReadBlocks(pSeid, od, blockNo, size, id);
  ASSERTIONS
    FCAopenTableExists(pSeid);
  EFFECTS
    rr = rr1;
    NOT d.addressable
      => 'FCAinputStream(fSeid)
            = VECTOR(FOR i FROM LENGTH(rr1.data) + 1
                           TO LENGTH(FCAinputStream(fSeid))
                       : FCAinputStream(fSeid)[i]);

OVFUN FCAwriteBlocks(seid pSeid; openDescriptor od; INTEGER blockNo;
                    VECTOR_OF fileBlock vfb; asyncId id)
  *> ioStatus ios;                                         $(FCAwriteBlocks)
  $(LR ** needs asynchronous I/O)
  $(writes the contents of a vector of file blocks onto the object mentioned.
    the data must correspond to the parameters of the openable object.)
  DEFINITIONS
    seid fSeid IS FCAopenEntry(pSeid, od).openSeid;
    secureEntityType type IS SENseidType(fSeid);
    deviceStruct d
      IS IF type = tDevice THEN deviceDataSeid(fSeid)
           ELSE IF type INSET {tFile, tExtent}
             THEN STRUCT(TRUE, 512, 32768, 512, FCAfileSize(fSeid))
           ELSE STRUCT(FALSE, 1, 255, 1, 0);
    INTEGER size IS LENGTH(vfb);
    INTEGER blockCount IS size/512;
    INTEGER blockPast IS (IF SENseidType(fSeid) = tDevice THEN
                             MIN({d.maxBlockNo, blockNo+blockCount})
                          ELSE
                          blockNo+blockCount);

  EXCEPTIONS
    XbadOd: NOT (FCAopenEntry(pSeid, od) ~= ?);
    XbadSize: LENGTH(vfb) = 0;
    XnotWritable: NOT omWrite INSET FCAopenEntry(pSeid, od).openMode;
    XnoSpace:  RESOURCE_ERROR;
```

```
$(  EXCEPTIONS AFTER THIS POINT ARE ACTUALLY HANDLED AS I/O ERRORS
       In addition, the following exceptions may not be treated as "pure"
       in the implementation, i.e., they may involve effects in the code.)
    XbadSize:
      NOT LENGTH(vfb) INSET {d.minRequest .. d.maxRequest}
        OR (LENGTH(vfb) MOD d.modSize) ~= 0;
    XbadBlockNo: NOT {blockNo ..blockNo + LENGTH(vfb) *1} SUBSET
                     {0 .. d.maxBlockNo};
    XendOfFile: d.addressable AND type ~= tFile
      AND blockNo + LENGTH(vfb) > d.maxBlockNo;
    XnoSpace: RESOURCE_ERROR;
  ASSERTIONS
    FORALL INTEGER i|vfb[i]~=?; INTEGER j | vfb[j]~=? :
        LENGTH(vfb[i]) = LENGTH(vfb[j]);

  EFFECTS
    ios = (SOME ioStatus iosl | TRUE);
        IF d.addressable
          THEN FORALL INTEGER i
                 : 'FCAfileData(fSeid, i)
                      = (IF NOT i INSET {blockNo .. blockPast * 1}
                           THEN FCAfileData(fSeid, i)
                           ELSE vfb[i * blockNo + 1])
          ELSE 'FCAoutputStream(fSeid)
                  = VECTOR(FOR i FROM 1
                                 TO LENGTH(FCAoutputStream(fSeid))+LENGTH(vfb)
                            : IF i <= LENGTH(FCAoutputStream(fSeid))
                                 THEN FCAoutputStream(fSeid)[i]
                                 ELSE
                           vfb[(i*LENGTH(FCAoutputStream(fSeid)))/LENGTH(vfb)]
                        [(i*LENGTH(FCAoutputStream(fSeid)))MOD LENGTH(vfb)]);


$(********************** general device manipulation *********************)

VFUN FCAvDeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
                        VECTOR_OF INTEGER args; asyncId id)
  *> ioStatus status;                                   $(FCAvDeviceFunction)
  $(the purpose of this function is to return the value that FCAdeviceFunction
    would return if it were executed in a given state)
  $(the specification of this function is device*dependent, but may be
    filled in at a later time)
  DEFINITIONS
    seid dSeid IS FCAopenEntry(pSeid, od).openSeid;
  EXCEPTIONS
    XnoFile: FCAopenEntry(pSeid, od) = ?;
    XbadDevice: NOT SENseidType(dSeid) INSET {tDevice, tTerminal};
$( there should be an exception here which uses a table of functions
   vs read/write permissions required to return XnotReadable or
   XnotWritable as required.  JN )
  ASSERTIONS
    FCAopenTableExists(pSeid);
  DERIVATION
    SOME ioStatus ios | TRUE;

OVFUN FCAdeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
                          VECTOR_OF INTEGER args; asyncId id)
  *> ioStatus status;                                  $(FCAdeviceFunction)
  $(the specification of this function is device*dependent, but may be
    filled in at a later time)
  EXCEPTIONS
    EXCEPTIONS_OF FCAvDeviceFunction(pSeid, od, f, args, id);
  ASSERTIONS
    FCAopenTableExists(pSeid);
  EFFECTS
    $(the state of the device somehow changes, and the result is returned via
      the io status)
    status = FCAvDeviceFunction(pSeid, od, f, args, id);
    TRUE;

OFUN FCAterminalLock(seid pSeid, devSeid);                  $(FCAterminalLock)
  $(sets the current terminal in the group to be "devSeid".  The requesting
    process must have the privilege to lock terminals)
$(      Actually, this functionality is now performed by the
        function SETPATH through K_device_function.  JN)
  EXCEPTIONS
    XnoPriv: NOT SMXhasPriv(pSeid, privSetPath);
    XnoClass: SENseidType(devSeid) ~= tTerminal;
    XnoFile: FCAinfo(devSeid) = ?;
  ASSERTIONS
    FCAopenTableExists(pSeid);
  EFFECTS
    LET terminalGroup t | devSeid INSET FCAterminalPathSet(t)
      IN 'FCAcurrentPath(t) = devSeid;

$(****************** mounting and unmounting operations ********************)

OVFUN FCAmount(seid pSeid, dev; BOOLEAN readOnly)     $(FCAmount)
        *> fileNsp newNsp;
  $(performs the logical mounting of a file system from an extent.  The
    semantics are described in the general commentary in the FCA
    specification)
  DEFINITIONS
    fileNsp openSlot IS
        SOME fileNsp freeMountNsp | FCAmountTable(freeMountNsp) = ?;
    fileSystem fileSys IS
      FCAextentToFileSys(VECTOR(FOR i FROM 1 TO FCAfileSize(dev)
                                : FCAfileData(dev, i)), openSlot) ;
      $(the file system produced by the data on the extent)
    fileSystemEntry fse(seid f) IS
      SOME fileSystemEntry fsel | fsel INSET fileSys AND fsel.fileSeid = f;
      $(the entry in the file system with a given seid)
  EXCEPTIONS
    XbadPriv: NOT SMXhasPriv(pSeid, privMount);
    XnoFile: NOT (FCAinfo(dev) ~= ?);
    XnoClass: SENseidType(dev) ~= tExtent;
    XmntSpace: RESOURCE_ERROR;
  ASSERTIONS
    FCAopenTableExists(pSeid);
  EFFECTS
    newNsp = openSlot;                  $( return new mount slot )
    'FCAmountTable(openSlot)
```

```
       = STRUCT(dev, readOnly, TIIinfo(dev), FCAinfo(dev));
    'FCAinfo(dev) = ?;
    FORALL INTEGER i : 'FCAfileData(dev, i) = ?;
    'TIIinfo(dev) = ?;
    FORALL seid f | fse(f) ~= ?
      : 'FCAinfo(f) = fse(f).gl
          AND (FORALL INTEGER i INSET {1 .. LENGTH(fse(f).fileData)}
                 : 'FCAfileData(f, i * 1) = fse(f).fileData[i])
          AND 'TIIinfo(f) = fse(f).tii;

OFUN FCAunmount(seid pSeid; fileNsp fsNsp);                           $(FCAunmount)
  $(performs logical unmounting of a file system.  Restores the extent
    so that it can be accessed)
  DEFINITIONS
    mountTableEntry mte IS FCAmountTable(fsNsp);
    fileSystem fs IS
      {fileSystemEntry fse
         | LET seid fSeid | SENseidNsp(fSeid) = fsNsp AND FCAinfo(fSeid) ~= ?
             IN fse = STRUCT(fSeid, FCAinfo(fSeid), TIIinfo(fSeid),
                             VECTOR(FOR i FROM 1 TO FCAfileSize(fSeid)
                                       : FCAfileData(fSeid, i * 1)))};
      $( the state of the file system to be unmounted)
    VECTOR_OF fileBlock extData
      IS SOME VECTOR_OF fileBlock vfb
           | FCAextentToFileSys(vfb,fsNsp) = fs;
       $(given the state of the file system, the extent that is equivalent to
         it)
    seid dev IS FCAmountTable(fsNsp).devSeid; $( underlying device )
  EXCEPTIONS
    XbadPriv: NOT SMXhasPriv(pSeid, privMount);
    XnoXXX: FCAmountTable(fsNsp) = ?;
    XnoTranquil:
      EXISTS seid fSeid | SENseidNsp(fSeid) = fsNsp
        : FCAinfo(fSeid).openCount > 0;
  ASSERTIONS
    FCAopenTableExists(pSeid);
  EFFECTS
    'FCAinfo(dev) = mte.devGl;
    'TIIinfo(dev) = mte.devTii;
    FORALL INTEGER i INSET {1 .. LENGTH(extData)}
      : 'FCAfileData(dev, i * 1) = extData[i];
    $(the device's state "comes back")
    FORALL seid fSeid | SENseidNsp(fSeid) = fsNsp
      : 'TIIinfo(fSeid) = ?
           AND 'FCAinfo(fSeid) = ?
           AND (FORALL INTEGER i : 'FCAfileData(fSeid, i) = ?);
    $(the state of all files on the file system "disappears")
    FCAmountTable(fsNsp) = ?;


END_MODULE
```

```
$("     MODULE:         lev.specs (version 4.9)
        CONTENTS:       System Levels
        TYPE:           SPECIAL specifications
        LAST CHANGED:   7/28/80, 09:51:43
  ")


MODULE lev


$( This module enables the centralization of the get and set level operations
   for all modules.  Each module maintains the type-independent operation
   of its own objects, and applies certain conditions to the getting and
   setting of this information.  However, there is only one kernel operation
   for getting levels, and one for setting levels, of all objects.  The
   operations are combined in this module)

    TYPES

    $(from smx)
nonDisType: STRUCT_OF(
               INTEGER securityLevel; SET_OF securityCat securityCatS;
               INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
daType: SET_OF daMode;
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
tiiStruct: STRUCT_OF(nonDisType nd; modeStruct ms; INTEGER owner, group;
                     SET_OF privType priv);

    $(FROM pvm)
globalData:
  STRUCT_OF(BOOLEAN sharable, swappable, sticky, memAdvise, executable;
            direction growth);
instanceStruct:
  STRUCT_OF(globalData gl; direction growth; INTEGER refCount; VECTOR_OF INTEGER data);

    $(FROM fca)
openFileEntry: STRUCT_OF(seid openSeid; SET_OF openModes openMode);


    EXTERNALREFS

    FROM smx:
seid: DESIGNATOR;
secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
                   tExtent, tNull};
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;
daMode: {daRead, daWrite, daExecute};
        privType: {
                          privFileUpdateStatus,  privLink,
                          privLockSeg,           privModifyPriv,
                          privMount,             privSetLevel,
                          privStickySeg,         privSetPath,
                          privViolSimpSecurity,  privViolStarSecurity,
                          privViolSimpIntegrity, privViolStarIntegrity,
                          privViolDiscrAccess,   privRealizeExecPermission,
                         privSignal,             privWalkPTable,
                         privHalt,               privKernelCall,
                         privViolCompartments,   privSetComm,
                         privImmigrate
                         };

VFUN SENseidType(seid s) *> secureEntityType set;
VFUN TIIgetEntityLevel(seid pSeid, oSeid) *> tiiStruct ntii;
OFUN TIIsetEntityLevel(seid pSeid, oSeid; tiiStruct ntii);

    FROM pvm:
direction: {up, down};
VFUN SEGinstanceInfo(seid s) *> instanceStruct is;

    FROM fca:
openDescriptor: DESIGNATOR;
openModes: {omRead, omWrite, omExclusiveRead, omExclusiveWrite};
VFUN FCAopenEntry(seid pSeid; openDescriptor od) *> openFileEntry oe;


    FUNCTIONS

VFUN LEVgetObjectLevel(seid pSeid, oSeid) *> tiiStruct otii;
  EXCEPTIONS
    EXCEPTIONS_OF TIIgetEntityLevel(pSeid, oSeid);
  DERIVATION
    TIIgetEntityLevel(pSeid, oSeid);

OFUN LEVsetObjectLevel(seid pSeid, oSeid; tiiStruct otii);
                                                        $(LEVsetObjectLevel)
  DEFINITIONS
    secureEntityType type IS SENseidType(oSeid);
  EXCEPTIONS
    EXCEPTIONS_OF TIIsetEntityLevel(pSeid, oSeid, otii);
    XnoObj :(type = tSegment
                  AND SEGinstanceInfo(oSeid).gl.sharable = TRUE)
   OR (type INSET {tFile, tDevice, tSubtype, tTerminal, tExtent}
        AND (EXISTS seid pSeid1; openDescriptor od :
                FCAopenEntry(pSeid1, od).openSeid = oSeid));
  EFFECTS
    EFFECTS_OF TIIsetEntityLevel(pSeid, oSeid, otii);


END_MODULE
```

```
mac.specs     Page 1    Wed Mar 18 11:12:12 1981

$("     MODULE:          mac.specs (version 4.5)
        CONTENTS:        Machine
        TYPE:            SPECIAL.specifications
        LAST CHANGED:    7/28/80, 09:52:15
  ")


MODULE mac

    PARAMETERS

INTEGER MACmaxVAddr, $( maximum virtual address, also maximum segment size
                2^16 - 1 on PDP-11/70)
        MACmaxOffset, $( maximum offset component of virtual address,
                         2^13 - 1 on  PDP-11/70)
        MACmaxReg; $( maximum memory mapping register address, seven on
                         PDP-11/70)

    FUNCTIONS

VFUN MACclock() -> INTEGER time;
  $( integer that represents real time)
  INITIALLY TRUE;

OFUN MACclockIncrement();
  $( invlked continuously by a separate abstract process -- the system clock)
  EFFECTS
    'MACclock() = MACclock() + 1;


END_MODULE
```

```
$("      MODULE:          pbl.specs (version 4.7)
         CONTENTS:        Parameter Block Functions
         TYPE:            SPECIAL specifications
         LAST CHANGED:    12/23/80, 17:17:46
  ")


MODULE pbl

$( This module specifies the action of getting arguments or putting results
   of operations into the virtual memory of a process.  The part of
   virtual memory so manipulated is called a parameter block.  The need
   for parameter blocks comes about when the length of the data is not
   constant for all invocations of a given operation.

   To make specifications more readable, all parameter block operations are
   specified in two parts.  The basic functionality of an operation is
   specified in the module to whose object the operation refers, e.g.,
   the basic specification of readBlock comes from the fmi module.  The
   parameter block manipulation, along with appropriate data conversion,
   is specified here.  This decomposition removes the issue of parameter
   blocks from the basic specification of already complicated operations.
   The parameter block manipulation becomes simple once isolated here.)


    TYPES

    $(from mac)
vAddrType: {0 .. MACmaxVAddr};

    $(from pvm)
virtualLocation:
  STRUCT_OF(domainType domain; spaceType idSpace; vAddrType vAddr);
pBlock: STRUCT_OF(virtualLocation vloc; INTEGER size);

    $(from fca)
asyncId: CHAR;
fileStatus: STRUCT_OF(INTEGER nBlocks, linkCount, timeLastMod; seid subtype;
                      BOOLEAN openAtCrash);
ioStatus: STRUCT_OF(INTEGER devDep);
fileBlock: VECTOR_OF CHAR;
readResult: STRUCT_OF(VECTOR_OF fileBlock data; ioStatus errst);

    $(from spf)
SPFargs: VECTOR_OF INTEGER;


    EXTERNALREFS

    FROM mac:
INTEGER MACmaxVAddr;

    FROM smx:
seid: DESIGNATOR;
domainType: {nullDomain, kernelDomain,supervisorDomain,userDomain};

```

```
    FROM pvm:
spaceType: {nullSpace,iSpace, dSpace};
OFUN PVMstore(seid pSeid; pBlock block; VECTOR_OF INTEGER vec);
VFUN PVMretrieve(seid pSeid; pBlock block) *> VECTOR_OF INTEGER vec;

    FROM fca:
openDescriptor: DESIGNATOR;
IOfunction: {rewind, etc};

    FROM fmi:
VFUN FCAvDeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
                        VECTOR_OF INTEGER args; asyncId id)
  *> ioStatus status;
OVFUN FCAdeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
                        VECTOR_OF INTEGER args; asyncId id)
  *> ioStatus status;
VFUN FCAvReadBlocks(seid pSeid; openDescriptor od; INTEGER blockNo, size;
                    asyncId as) *> readResult rr;
OVFUN FCAreadBlocks(seid pSeid; openDescriptor od; INTEGER blockNo, size;
                    asyncId as) *> readResult rr;
OVFUN FCAwriteBlocks(seid pSeid; openDescriptor od; INTEGER blockNo;
                     VECTOR_OF fileBlock vfb; asyncId id) *> ioStatus ios;

    FROM spf:
SPFfunctionType: {syncSPF, immSegSPF, sysHaltSPF, levelSetSPF};
OFUN SPFspecialFunction(seid pSeid; SPFfunctionType fn; SPFargs args);


    ASSERTIONS

FORALL VECTOR_OF fileBlock vfb
  : PBLwordsToBlocks(PBLblocksToWords(vfb)) = vfb;


    FUNCTIONS

$(************************* data conversion functions *********************)

VFUN PBLioStatToVec(ioStatus ios) *> VECTOR_OF INTEGER vi;
  HIDDEN;
  INITIALLY vi ~= ?;

VFUN PBLblocksToWords(VECTOR_OF fileBlock vfb) *> VECTOR_OF INTEGER vi;
  HIDDEN;
  INITIALLY vi ~= ?;

VFUN PBLwordsToBlocks(VECTOR_OF INTEGER vi) *> VECTOR_OF fileBlock vfb;
  HIDDEN;
  DERIVATION SOME VECTOR_OF fileBlock vfbl | PBLblocksToWords(vfbl) = vi;

$(***************************** operations ********************************)

OFUN PBLdeviceFunction(seid pSeid; openDescriptor od; IOfunction f;
                       pBlock arguments, status; asyncId id);
                                                   $(PBLdeviceFunction)
  DEFINITIONS
```

```
    VECTOR_OF INTEGER inargs IS PVMretrieve(pSeid, arguments);
    ioStatus st IS FCAvDeviceFunction(pSeid, od, f, inargs, id);
    VECTOR_OF INTEGER result IS PBLioStatToVec(st);
  EXCEPTIONS
    EXCEPTIONS_OF PVMretrieve(pSeid, arguments);
    EXCEPTIONS_OF FCAdeviceFunction(pSeid, od, f, inargs, id);
    EXCEPTIONS_OF PVMstore(pSeid, status, result);
  EFFECTS
    EFFECTS_OF PVMstore(pSeid, status, result);
    st = EFFECTS_OF FCAdeviceFunction(pSeid, od, f, inargs, id);

OVFUN PBLreadBlock(seid pSeid; openDescriptor od; INTEGER blockNo;
                   pBlock duFile; asyncId as)
    => STRUCT_OF(INTEGER bytesRead; ioStatus errst) result;   $(PBLreadBlock)
  DEFINITIONS
    readResult rr IS FCAvReadBlocks(pSeid, od, blockNo, duFile.size, as);
    VECTOR_OF INTEGER intData IS PBLblocksToWords(rr.data);
  EXCEPTIONS
    EXCEPTIONS_OF FCAreadBlocks(pSeid, od, blockNo, duFile.size, as);
    EXCEPTIONS_OF PVMstore(pSeid, duFile, intData);
  EFFECTS
    result = STRUCT(LENGTH(intData), rr.errst);
    EFFECTS_OF PVMstore(pSeid, duFile, intData);

OFUN PBLspecialFunction(seid pSeid; SPFfunctionType fn; pBlock parm);
                                                         $(PBLspecialFunction)
  DEFINITIONS
    SPFargs args IS PVMretrieve(pSeid, parm);
  EXCEPTIONS
    EXCEPTIONS_OF PVMretrieve(pSeid, parm);
    EXCEPTIONS_OF SPFspecialFunction(pSeid, fn, args);
  EFFECTS
    EFFECTS_OF SPFspecialFunction(pSeid, fn, args);

OVFUN PBLwriteBlock(seid pSeid; openDescriptor od; INTEGER blockNo;
                    pBlock duFile; asyncId id) => ioStatus ios;
                                                              $(PBLwriteBlock)
  DEFINITIONS
    VECTOR_OF fileBlock vfb IS PBLwordsToBlocks(PVMretrieve(pSeid, duFile));
  EXCEPTIONS
    EXCEPTIONS_OF PVMretrieve(pSeid, duFile);
    EXCEPTIONS_OF FCAwriteBlocks(pSeid, od, blockNo, vfb, id);
  EFFECTS
    ios = EFFECTS_OF FCAwriteBlocks(pSeid, od, blockNo, vfb, id);

END_MODULE
```

```
$("      MODULE:          pro.specs (version 4.11)
         CONTENTS:        Process Operators
         TYPE:            SPECIAL.specifications
         LAST CHANGED:    12/23/80, 16:56:41
  ")


MODULE pro

$( this module now contains the material which was formerly in
   the pro, ipc and pst modules)

    TYPES

    $(types supporting pseudo interrupts)
piLevelType: {PROminPiLevel .. PROmaxPiLevel}; $(pseudo interrupt level range)
piEntryType: STRUCT_OF(piLevelType oldPil;
                       INTEGER oldPc;
                       INTEGER oldPs;
                       ipcMessageType parameter;
                       INTEGER newPc;
                       INTEGER newPs);
piVectorType: {VECTOR_OF piEntryType piv |
                            LENGTH(piv) = PROmaxPiLevel-PROminPiLevel+1};

    $(types supporting ipc)
ipcqType: {VECTOR_OF ipcMessageType zz | LENGTH(zz) <= IPCmaxMessageCount};
ipcTextType: {VECTOR_OF CHAR vc | LENGTH(vc) = IPCmaxMessageLength};
ipcMessageType: STRUCT_OF(seid sender; ipcTextType text);
pendingType: STRUCT_OF(BOOLEAN flag; INTEGER time);

    $(structure of process status information)
processStatusType: STRUCT_OF(seid self;  $(part of the process state that users can state)
                              seid parent;
                           INTEGER family;
                           INTEGER realUser;
                           INTEGER realGroup;
                           INTEGER advPrio;          $(advisory priority)
                           piLevelType pil;
                           INTEGER ps;
                           INTEGER timerAlarm;
                           INTEGER clock;
                          BOOLEAN timTog;
                          INTEGER supervisorTiming;
                          INTEGER userTiming);
processStateType: STRUCT_OF(INTEGER pc; $(program counter)
                               VECTOR_OF BOOLEAN vpend;
                                piVectorType piv;
                                ipcqType ipcq;
                                processStatusType pStat);
    $(from smx)
nonDisType: STRUCT_OF(
                INTEGER securityLevel; SET_OF securityCat securityCatS;
                INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
daType: SET_OF daMode;
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
```

```
tiiStruct: STRUCT_OF(nonDisType nd;
               modeStruct ms; INTEGER owner, group; SET_OF privType priv);


    PARAMETERS

INTEGER PROmaxProcessCount;

INTEGER PROminPiLevel; $(most interruptable pseudo interrupt level)
INTEGER PROmaxPiLevel; $(least interruptable pseudo interrupt level)
INTEGER piZero,piIPC,piTimer,piSignal,piIOC,piHardwareFault;
  $(defined pseudo interrupt levels,I/O completion pseudo'interupt)

INTEGER IPCmaxMessageCount; $(maximum number of ipc messages per process)
INTEGER IPCmaxMessageLength; $(maximum number of characters per ipc message)
INTEGER piPc, $(program counter for process invocation and spawning=0)
        piPs; $(processor state for process invocation and spawning,
                 154000 octal = user previous domain, supervisor domain,alternate register

seid processExampleSeid; $(any seid with the tProcess nsp)
INTEGER SignalTimeOut;

    DEFINITIONS

BOOLEAN processExists(seid pSeid) IS PSTprocessState(pSeid) ~= ?;
seid newProcessSeid IS $(the process seid generation algorithm)
  SENmakeSeid(processExampleSeid, SENseidIndex(
                                      SOME seid s |
                                        SENseidType(s) = tProcess
                                        AND NOT processExists(s)));
INTEGER processCount IS
  CARDINALITY({INTEGER i | PSTprocessSlot(i) ~= ?});
INTEGER emptySlot IS SOME INTEGER i | i INSET {1 .. PROmaxProcessCount}
                                       AND PSTprocessSlot(i) = ?;


    EXTERNALREFS

    FROM mac:
VFUN MACclock() -> INTEGER time;

    FROM smx:
seid: DESIGNATOR;
secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment,
                   tSubtype, tExtent, tNull};
        privType: {
                           privFileUpdateStatus,  privLink,
                           privLockSeg,           privModifyPriv,
                           privMount,             privSetLevel,
                           privStickySeg,         privSetPath,
                           privViolSimpSecurity,  privViolStarSecurity,
                           privViolSimpIntegrity, privViolStarIntegrity,
                           privViolDiscrAccess,   privRealizeExecPermission,
                           privSignal,            privWalkPTable,
                           privHalt,              privKernelCall,
                           privViolCompartments,  privSetComm,
                              privImmigrate
                              };

daMode: {daRead, daWrite, daExecute};
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;
VFUN SENseidIndex(seid s) -> INTEGER index;
VFUN SENseidType(seid s) -> secureEntityType t;
VFUN SENmakeSeid(seid exampleSeid; INTEGER index) -> seid rSeid;
VFUN SMXhasPriv(seid pSeid; privType priv) -> BOOLEAN b;
VFUN SMXflow(seid pSeid, oSeid; daType d) -> BOOLEAN b;
VFUN SMXdap(seid pSeid, oSeid, daType d) -> BOOLEAN b;
VFUN TIIgetEntityLevel(seid pSeid, oSeid) ->tiiStruct otii;
VFUN TIIinfo(seid s) -> tiiStruct tiis;
OFUN TIIsetEntityLevel(seid pSeid, oSied; tiiStruct ntii);

    FROM pvm:
segDes: DESIGNATOR;

    FROM pvp:
OFUN PVPforkSupport(seid parent, child);
OFUN PVPinvokeSupport(seid pSeid, immSeid; segDes arg);
OFUN PVPspawnSupport(seid parent, child, imSeid; segDes arg);
OFUN PVPreleaseProcessSupport(seid pSeid);

    FROM fca:
OFUN FCAcreateOpenTable(seid pSeid);       $(spawn support)

    FROM fmi:
OFUN FMIforkSupport(seid parent, child);
OFUN FMIreleaseSupport(seid pSeid);


    ASSERTIONS
PROminPiLevel <= piZero; piZero < piIPC; piIPC < piTimer; piTimer < piSignal; piSignal < p
piIOC < piHardwareFault; piHardwareFault <= PROmaxPiLevel;
  $( ordering of defined pseudo interrupt levels)
SENseidType(processExampleSeid) = tProcess;
  $(processExampleSeid is an example of a process seid)
processCount <= PROmaxProcessCount; $(there are never too many processes)
FORALL seid s: SMXflow(s, processExampleSeid, {daRead});
  $(the processExampleSeid can be referenced from any security level)


    FUNCTIONS

$( **************** process state functions *************** )

VFUN PSTprocessState(seid pSeid) -> processStateType ps;
  HIDDEN;
  INITIALLY ps = ?;

VFUN PSTprocessStatus(seid pSeid) -> processStatusType ps;
  HIDDEN;
  DERIVATION PSTprocessState(pSeid).pStat;

```

```
VFUN PSTprocessSlot(INTEGER n) *> seid ps;
 HIDDEN;
 INITIALLY ps = ?;

$( **************support for K_walk_process_table **************)

VFUN PROwalkProcessTable(seid pSeid; INTEGER n) *> seid rSeid;
 EXCEPTIONS
  XnoPriv: NOT privWalkPTable INSET TIIinfo(pSeid).priv;
  XbadSlot: NOT n+1 INSET {1 .. PROmaxProcessCount};
  Xempty:PSTprocessSlot(n) = ?;
 DERIVATION
  PSTprocessSlot(n);

$( ************** support for K_nap **************)

VFUN PSTtimeOfNap(seid pSeid) *> INTEGER time;
 HIDDEN;
 INITIALLY time = 0;

OFUN PROnap(seid pSeid; INTEGER timeout);
 DELAY WITH 'PSTtimeOfNap(pSeid) = MACclock();
    UNTIL MACclock() >= PSTtimeOfNap(pSeid) + timeout
       OR PSTprocessStatus(pSeid).pil > piIPC;
 $(Wakeup on pseudo interrupt occurs when it becomes pending.
 After waking, the pseudo interrupt is asserted.)

$( ****support for pseudo interrupts: K_signal and K_interruptReturn****)

VFUN PSTtimeOfSignal(seid pSeid) *> INTEGER time;
HIDDEN;
INITIALLY time = 0;

OFUN PROsignal(seid sender, receiver; ipcTextType signalMsg);
 DEFINITIONS
  piEntryType receiversSignalEntry()
   IS PSTprocessState(receiver).piv[piSignal];
 EXCEPTIONS
  XnoPriv: NOT SMXhasPriv(sender, privSignal);
  XbPrSd: NOT (processExists(receiver)
            AND SMXflow(sender, receiver, {daRead, daWrite}));
   XunInt:
             NOT PSTprocessState(receiver).pStat.pil > piSignal;
DELAY WITH 'PSTtimeOfSignal(sender) = MACclock();
UNTIL MACclock() >= PSTtimeOfSignal(sender) + SignalTimeOut
  OR PSTprocessState(receiver).pStat.pil <= piSignal;
 ASSERTIONS
  processExists(sender);
 EFFECTS
  'PSTprocessState(receiver).vpend[piSignal] = TRUE;
  'PSTprocessState(receiver).piv[piSignal].parameter.text = signalMsg;
  'PSTprocessState(receiver).piv[piSignal].parameter.sender = sender;
$(PROBLEMS
  Design requires that K_signal interrupt long kernel calls. How is this
  this to be done?)
    $("the only Kcalls which will be interrupted are K*nap,
             K_receive (time out) and a synchronous K_read_block
        from the terminal. At least for the K_nap and K_receive
        delays, this can be specified via another condition in the
        PROnap and PROreceive")

OFUN PROinterruptReturn(seid pSeid);
$(emulates rti instruction. Uses pseudo interrupt vector entry associated
  with current level and restores pc, ps, and pil to their "old" values)
  DEFINITIONS
    piEntryType currentInterruptEntry
      IS PSTprocessState(pSeid).piv[PSTprocessState(pSeid).pStat.pil];
  EFFECTS
    'PSTprocessState(pSeid).pc = currentInterruptEntry.oldPc;
    'PSTprocessState(pSeid).pStat.ps = currentInterruptEntry.oldPs;
    'PSTprocessState(pSeid).pStat.pil = currentInterruptEntry.oldPil;

$( ********** support for ipc: K_post and K_receive **********)

OFUN PROpost(seid sender,receiver; BOOLEAN postPseudoInterrupt;
             ipcTextType text);
$(Detects security violation and overflow. Appends message to the tail of
  the receivers ipcq. Posts pseudo interrupt in the receiver if requested
  and if receiver has no pending receive)
  DEFINITIONS
    VECTOR_OF ipcMessageType queue() IS PSTprocessState(receiver).ipcq;
    INTEGER qLength IS LENGTH(queue());
$(There ARE exceptions in this function, but writing up is permitted
  and the presence or absence of an exception might give information about
  the state of a write-only object,NOTE THIS AND FIX IT)
  ASSERTIONS
    processExists(sender);
  EFFECTS
    processExists(receiver)
        AND SMXflow(sender, receiver, {daWrite})
        AND qLength < IPCmaxMessageLength
        AND SMXdap(sender,receiver, {daWrite})
      => 'PSTprocessState(receiver).ipcq
             = VECTOR(FOR i FROM 1 TO MIN({1+qLength, IPCmaxMessageLength})
                        : IF i <= qLength
                            THEN queue()[i]
                            ELSE STRUCT(sender, text));
               $(append new message to receivers queue)
           (postPseudoInterrupt
                AND NOT PSTreceivePending(receiver).flag
              => 'PSTprocessState(receiver).vpend[piIPC] = TRUE);
             $(post ipc pseudo interrupt if required and if receiver has
               no pending read)

VFUN PSTreceivePending(seid pSeid) -> pendingType r;
  HIDDEN;
  INITIALLY r.flag = FALSE AND r.time = 0;

OVFUN PROreceive(seid pSeid; INTEGER timeout) -> ipcMessageType msg;
$(Returns the ipc message at the head of the queue if one exists or arrives
  before the expiration of timeout. Otherwise it returns an exception)
  DEFINITIONS
```

```
    VECTOR_OF ipcMessageType queue() IS PSTprocessState(pSeid).ipcq;
    INTEGER qLength IS LENGTH(queue());
    pendingType pending() IS PSTreceivePending(pSeid);
EXCEPTIONS
XbNoMes: qLength = 0;
  DELAY WITH 'PSTreceivePending(pSeid).flag = TRUE;
             'PSTreceivePending(pSeid).time = MACclock();
      UNTIL qLength > 0 OR MACclock() >= pending().time + timeout;
  EFFECTS
    IF qLength > 0 THEN msg =  queue()[1] ELSE TRUE;
    'PSTprocessState(pSeid).ipcq = VECTOR( FOR i FROM 1 TO qLength*1 : queue()[i+1]);
    'PSTreceivePending(pSeid).flag = FALSE;

$( ********* support for K_fork **********)

OVFUN PROfork(seid parent) *> seid child;
  DEFINITIONS
    processStateType p IS PSTprocessState(parent);
    seid c IS newProcessSeid; processStatusType s IS p.pStat;
  EXCEPTIONS
    XnoProc: processCount+1 > PROmaxProcessCount;
    EXCEPTIONS_OF FMIforkSupport(parent, c);
    EXCEPTIONS_OF PVPforkSupport(parent, c);
  EFFECTS
    child = c;
    'PSTprocessSlot(emptySlot) = c;
    'PSTprocessState(c) = STRUCT(p.pc,
                                 VECTOR(),
                                 VECTOR(),
                                 p.ipcq,
                                 STRUCT(c,
                                        parent,
                                        s.family,
                                        s.realUser,
                                        s.realGroup,
                                        s.advPrio,
                                        PROmaxPiLevel,
                                        s.ps,
                                        0,
                                        ?,
                                        FALSE,
                                        s.supervisorTiming,
                                        s.userTiming));
    $(this assertion specifies the initial tdi state of a forked child)
    'TIIinfo(c) = TIIinfo(parent);
    $(this assertion specifies the initial tii state of a forked child)
    EFFECTS_OF FMIforkSupport(parent, c); $(provide and copy pofv)
    EFFECTS_OF PVPforkSupport(parent, c); $(provide virtual memory)

$( ********** support for K_invoke **********)

OFUN PROinvoke(seid pSeid, immSeid; segDes arg);
  DEFINITIONS
    processStateType p IS PSTprocessState(pSeid);
    processStatusType s IS p.pStat;
    tiiStruct pt IS TIIinfo(pSeid);
  EXCEPTIONS
    EXCEPTIONS_OF PVPinvokeSupport(pSeid, immSeid, arg);
  ASSERTIONS
    processExists(pSeid);
  EFFECTS
    'PSTprocessState(pSeid) = STRUCT(piPc,
                              VECTOR(),
                              VECTOR(),
                              VECTOR(),
                              STRUCT(s.self,
                                     s.parent,
                                     s.family,
                                     s.realUser,
                                     s.realGroup,
                                     s.advPrio,
                                     PROmaxPiLevel,
                                     piPs,
                                     0,
                                     ?,
                                     FALSE,
                                     0,
                                     0));
  $(assertion defines thes initial process state of the invoked intermediary)
    'TIIinfo(pSeid) = STRUCT(pt.nd, pt.ms, pt.owner, pt.group,
                             TIIinfo(immSeid).priv);
    $(this is how the post*invoke process gets the intermeds privileges)
    EFFECTS_OF PVPinvokeSupport(pSeid, immSeid, arg); $(redo virtual memory)

$( ************** support for K_spawn **************)

OVFUN PROspawn(seid parent, immSeid; segDes arg) *> seid child;
  DEFINITIONS
    processStateType p IS PSTprocessState(parent);
    processStatusType s IS p.pStat;
    tiiStruct pt IS TIIinfo(parent);
    seid c IS newProcessSeid;
  EXCEPTIONS
    XnoProc: processCount+1 > PROmaxProcessCount;
    EXCEPTIONS_OF PVPspawnSupport(parent, c, immSeid, arg);
    EXCEPTIONS_OF FCAcreateOpenTable(parent);
  ASSERTIONS
    processExists(parent);
  EFFECTS
    child = c;
    'PSTprocessSlot(emptySlot) = c;
    'PSTprocessState(c) =  STRUCT(piPc,
                              VECTOR(),
                              VECTOR(),
                              VECTOR(),
                              STRUCT(c,
                                     s.parent,
                                     s.family,
                                     s.realUser,
                                     s.realGroup,
                                     s.advPrio,
                                     PROmaxPiLevel,
                                     piPs,
                                     0,
                                     ?,
                                     FALSE,
                                     0,
                                     0));
    $(the process state of the newly spawned intermediary)
    'TIIinfo(c) = STRUCT(pt.nd, pt.ms, pt.owner, pt.group,
                             TIIinfo(immSeid).priv);
    $(post*spawn child acquires intermediatarys privileges)
    EFFECTS_OF PVPspawnSupport(parent, c, immSeid, arg); $(create vm)
    EFFECTS_OF FCAcreateOpenTable(c); $(create pofv)

$( ************** support for K_release_process ****************)

OFUN PROreleaseProcess(seid pSeid, rSeid);
$(Typically a process will release itself and pSeid=rSeid. However this
is not treated as a special case.)
  EXCEPTIONS
    XbPrSd: (NOT processExists(rSeid))
       OR (NOT SMXflow(pSeid, rSeid, {daRead, daWrite}))
       OR ((TIIinfo(rSeid).owner ~= TIIinfo(pSeid).owner))
           AND NOT SMXhasPriv(pSeid, privSetLevel);
  ASSERTIONS
    processExists(pSeid);
  EFFECTS
    'PSTprocessSlot(SOME INTEGER i | PSTprocessSlot(i) = rSeid) = ?;
    'PSTprocessState(rSeid) = ?;
    'TIIinfo(rSeid) = ?;
    EFFECTS_OF FMIreleaseSupport(rSeid);
    EFFECTS_OF PVPreleaseProcessSupport(rSeid);

$( *************** status getting and setting ***************)

VFUN PROgetProcessStatus(seid pSeid, oSeid) *> processStatusType ps;
  DEFINITIONS
    processStatusType s IS PSTprocessStatus(oSeid);
  EXCEPTIONS
    XbPrSd: NOT (processExists(oSeid)
                         AND SMXflow(pSeid, oSeid, {daRead})
        AND SMXdap(pSeid,oSeid, {daRead}));
  ASSERTIONS
    processExists(pSeid);
  DERIVATION
    STRUCT(s.self,
           s.parent,
           s.family,
           s.realUser,
           s.realGroup,
           s.advPrio,
           s.pil,
           s.ps,
           s.timerAlarm,
           MACclock(),
           s.timTog,
           s.supervisorTiming,
           s.userTiming);

OFUN PROsetProcessStatus(seid pSeid, oSeid; processStatusType n);
  DEFINITIONS
    processStateType e IS PSTprocessState(oSeid);
    processStatusType o IS PSTprocessStatus(oSeid);
  EXCEPTIONS
    XbPrSd: (NOT processExists(oSeid))
       OR (NOT SMXflow(pSeid, oSeid, {daRead, daWrite}))
       OR ((TIIinfo(oSeid).owner ~= TIIinfo(pSeid).owner))
           AND NOT SMXhasPriv(pSeid, privSetLevel);
EXCEPTIONS_OF PROreleaseProcess(pSeid,oSeid);
  ASSERTIONS
    processExists(pSeid);
  EFFECTS
     'PSTprocessState(oSeid) = STRUCT(e.pc,
                                      e.vpend,
                                      e.piv,
                                      e.ipcq,
                                      STRUCT(o.self,
                                             o.parent,
                                             n.family,
                                             n.realUser,
                                             n.realGroup,
                                             n.advPrio,
                                             n.pil,
                                             n.ps,
                                             n.timerAlarm,
                                             ?,
                                             n.timTog,
                                             o.supervisorTiming,
                                             o.userTiming));
   END_MODULE
```

```
$("     MODULE:          pvm.specs (version 4.11)
        CONTENTS:        Virtual Memory
        TYPE:            SPECIAL.specifications
        LAST CHANGED:    12/23/80, 17:37:23
  ")


MODULE pvm

$( this module now contains the contents of what was the seg and pvm modules)

    TYPES

    $(FROM mac)
vAddrType: {0 .. MACmaxVaddr};
    $(FROM smx '' tii)
daType: SET_OF daMode;
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
domainType: {nullDomain,kernelDomain,supervisorDomain,userDomain};
nonDisType: STRUCT_OF(
                INTEGER securityLevel; SET_OF securityCat securityCatS;
                INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
tiiStruct: STRUCT_OF(nonDisType nd;
                modeStruct ms; INTEGER owner, group; SET_OF privType priv);
    $(from seg '' exportable)
segDes: DESIGNATOR;
spaceType: {nullSpace, iSpace, dSpace};
direction: {up, down};
    $(from seg '' redeclarable)
virtualLocation:
  STRUCT_OF(domainType domain; spaceType idSpace; vAddrType vAddr);
    $(from seg '' local)
globalData:
  STRUCT_OF(BOOLEAN sharable, swappable, sticky, memAdvise, executable);
instanceStruct:
  STRUCT_OF(globalData gl; direction growth; INTEGER refCount;
              VECTOR_OF INTEGER data);
useStruct: STRUCT_OF(seid instance; virtualLocation vloc; daType da);
    $(from pvm '' redeclarable)
pBlock: STRUCT_OF(virtualLocation vloc; INTEGER size);
statusStruct:
  STRUCT_OF(globalData gl; direction growth; INTEGER size; seid iseid;
                 segDes udes; virtualLocation vloc; daType da; BOOLEAN mapped);


    PARAMETERS

    $(from seg)seid exampleSegmentSeid; $(used for segment creation)
segDes SEGnullSeg; $( indicates null segment designator)
INTEGER PVMmaxSegDes; $(maximum number of segment designators in an address
                         space)


    DEFINITIONS

    $(from seg)
```

```
INTEGER SEGsize(seid segSeid) IS LENGTH(SEGinstanceInfo(segSeid).data);

    $(from pvm)
INTEGER nSegs(seid pSeid)
  IS CARDINALITY({segDes sd | SEGuseInfo(pSeid, sd) ~= ?});

BOOLEAN PVMvmExists(seid pSeid) IS PVMsegmentSet(pSeid) ~= ?;

segDes PVMblockToSeg(seid pSeid; pBlock block) IS
  SOME segDes sd
    | sd INSET PVMmappedSegmentSet(pSeid)
        AND (EXISTS useStruct use = SEGuseInfo(pSeid, sd)
               : use.vloc.domain = block.vloc.domain
                 AND use.vloc.idSpace = block.vloc.idSpace
                 AND use.vloc.vAddr <= block.vloc.vAddr
                 AND block.vloc.vAddr + block.size
                       <= use.vloc.vAddr + SEGsize(use.instance));
  $( gives the segment designator, if any, that totally contains block in
     address space designated by pSeid; if there is none, returns ?; the
     segment must be mapped)

SET_OF INTEGER addrRegRange(INTEGER vAddr, size; direction d) IS
  IF d  = up
    THEN {vAddr / MACmaxOffset .. (vAddr + size - 1) / MACmaxOffset}
    ELSE {(vAddr - size + 1) / MACmaxOffset .. vAddr / MACmaxOffset};
  $( gives the range of address registers used by a segment as a function of
     its start address, size, and growth direction)

SET_OF INTEGER addrRegRangeSeg(seid pSeid; segDes s) IS
  LET useStruct use = SEGuseInfo(pSeid, s)
    IN addrRegRange(use.vloc.vAddr, SEGsize(use.instance),
                    SEGinstanceInfo(use.instance).growth);
  $( gives the range of address registers used by a segment as a function of
     the process id and the segment designator)

BOOLEAN noHole(seid pSeid; INTEGER size; virtualLocation vl; direction d;
               SET_OF segDes ssd) IS
  NOT addrRegRange(vl.vAddr, size, d) SUBSET {0 .. MACmaxReg}
    OR (EXISTS segDes s | s INSET ssd; useStruct use = SEGuseInfo(pSeid, s)
          : use.vloc.idSpace = vl.idSpace
              AND use.vloc.domain = vl.domain
              AND addrRegRangeSeg(pSeid, s)
                      INTER addrRegRange(vl.vAddr, size, d) ~= {});
  $(TRUE iff a segment described by size, vl, and direction will NOT fit into
    a hole in the address space designated by pSeid and ssd; this includes
    testing for virtual memory underflow and overflow)

    EXTERNALREFS

    FROM mac:
INTEGER MACmaxVaddr, MACmaxOffset, MACmaxReg;

    FROM smx:
    $(FROM sen:)
seid: DESIGNATOR;
secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
                      tExtent, tNull};
VFUN SENseidNsp(seid s) -> INTEGER nsp;
VFUN SENseidIndex(seid s) -> INTEGER index;
VFUN SENmakeSeid(seid exampleSeid; INTEGER index) -> seid outSeid;
VFUN SENseidType(seid s) -> secureEntityType set;
    $(FROM prv:)
        privType: {
                             privFileUpdateStatus,  privLink,
                             privLockSeg,           privModifyPriv,
                             privMount,             privSetLevel,
                             privStickySeg,         privSetPath,
                             privViolSimpSecurity,  privViolStarSecurity,
                             privViolSimpIntegrity, privViolStarIntegrity,
                             privViolDiscrAccess,   privRealizeExecPermission,
                             privSignal,            privWalkPTable,
                             privHalt,              privKernelCall,
                             privViolCompartments,  privSetComm,
                             privImmigrate
                             };
    $(FROM tii:)
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;
daMode: {daRead, daWrite, daExecute};
VFUN TIIinfo(seid anySeid) -> tiiStruct tiiSt;
    $(FROM smx:)
VFUN SMXhasPriv(seid pSeid; privType priv) -> BOOLEAN b;
VFUN SMXflow(seid pSeid, oSeid; daType d) -> BOOLEAN b;
VFUN SMXdap(seid pSeid, oSeid; daType d) -> BOOLEAN b;


    ASSERTIONS
    $(from seg)
MACmaxVaddr > 0;  MACmaxOffset > 0; MACmaxReg > 0;
MACmaxVaddr + 1 = (MACmaxOffset + 1) * (MACmaxReg + 1);
PVMmaxSegDes >= 2;
   $( basic relations among the SEG parameters)
SENseidType(exampleSegmentSeid) = tSegment;
   $( basic property of exampleSegmentSeid and all segment seids)
FORALL seid s | SEGinstanceInfo(s) ~= ?
  : SENseidNsp(s) = SENseidNsp(exampleSegmentSeid);
   $(all seids for existing segments have a distinbuished nsp component)
FORALL seid s | SEGinstanceInfo(s) ~= ? : TIIinfo(s) ~= ?;
   $(all existing segments have an exiting TII entry)
FORALL seid pSeid; segDes segd | SEGuseInfo(pSeid, segd) ~= ?
  : SEGinstanceInfo(SEGuseInfo(pSeid, segd).instance) ~= ?;
   $(all valid segment uses have corresponding valid segment instances)

   $(from pvm)
FORALL seid pSeid | PVMvmExists(pSeid); segDes sd
  : (sd INSET PVMsegmentSet(pSeid)) = (SEGuseInfo(pSeid, sd) ~= ?);
   $(defines what it means for a segment tc be in the segment set of a
     process)
FORALL seid pSeid | PVMvmExists(pSeid)
  : PVMmappedSegmentSet(pSeid) SUBSET PVMsegmentSet(pSeid);
   $(only existing segments can be mapped)
FORALL seid pSeid1,pSeid2; segDes sd1,sd2:
  NOT SEGinstanceInfo(SEGuseInfo(pSeid1,sd1).instance).gl.sharable
  =>
  (SEGuseInfo(pSeid1,sd1).instance ~= SEGuseInfo(pSeid2,sd2).instance);
  $(A nonsharable instance struct cannot be pointed to by more than 1
  use struct.   Checked for in PVMrendezvous.)
FORALL seid pSeid | PVMvmExists(pSeid); segDes s1; segDes s2
  : LET useStruct use1 = SEGuseInfo(pSeid, s1);
        useStruct use2 = SEGuseInfo(pSeid, s2)
      IN s1 ~= s2 AND use1 ~= ? AND use2 ~= ?
          AND {s1, s2} SUBSET PVMmappedSegmentSet(pSeid)
          AND use1.vloc.domain = use2.vloc.domain
          AND use1.vloc.idSpace = use2.vloc.idSpace
        => addrRegRangeSeg(pSeid,  s1) INTER addrRegRangeSeg(pSeid, s2) = {};
   $(no two mapped segments in the same domain and idSpace may have
     overlapping memory address registers)


    FUNCTIONS

$(************************** state functions *****************************)

VFUN SEGinUseIndexSet() *> SET_OF INTEGER si;             $(SEGinUseIndexSet)
  $( gives the set of seid indices that are currently in use for existing
     segments)
  HIDDEN;
  INITIALLY si = {};

VFUN SEGinstanceInfo(seid segSeid) *> instanceStruct is;   $(SEGinstanceInfo)
  $( gives all the information pertaining to a segment's global data,
     referred to as segment in tance data)
  HIDDEN;
  INITIALLY is = ?;

VFUN SEGuseInfo(seid pSeid; segDes segd) *> useStruct us;       $(SEGuseInfo)
  $( gives all the information pertaining to a segment's use in the address
     space in a particular process; this is information local to a process)
  HIDDEN;
  INITIALLY us = ?;

VFUN PVMsegmentSet(seid pSeid) *> SET_OF segDes segSet;     $(PVMsegmentSet)
  $( gives the set of segments possessed by a given process)
  INITIALLY segSet = ?;

VFUN PVMmappedSegmentSet(seid pSeid) *> SET_OF segDes mappedSet;
                                                       $(PVMmappedSegmentSet)
  $( gives the set of mapped ** or active segments ** of a process; a
     segment cannot be addressed unless it is mapped)
  HIDDEN;
  INITIALLY mappedSet = ?;

$(********************* virtual memory management ************************)

OFUN PVMcreateVM(seid pSeid);                                   $(PVMcreateVM)
  $( creates a new virtual memory to be identified by "pSeid";  this VM
     must not currently exist)
  ASSERTIONS
```

```
    NOT PVMvmExists(pSeid);
  EFFECTS
    'PVMsegmentSet(pSeid) = {};
    'PVMmappedSegmentSet(pSeid) = {};

OFUN PVMdeleteVM(seid pSeid);                                  $(PVMdeleteVM)
  $( deletes the currently existing virtual memory "pSeid")
  ASSERTIONS
    PVMvmExists(pSeid);
  EFFECTS
    'PVMsegmentSet(pSeid) = ?;
    'PVMmappedSegmentSet(pSeid) = ?;

$(-------------- basic segment management --------------------)

OFUN PVMstore(seid pSeid; pBlock block; VECTOR_OF INTEGER vec);   $(PVMstore)
  $(inserts contents of vec into the mapped segment indicated by block)
  DEFINITIONS
    segDes targ IS PVMblockToSeg(pSeid, block);
    useStruct use IS SEGuseInfo(pSeid, targ);
    instanceStruct inst IS SEGinstanceInfo(use.instance);
  EXCEPTIONS
    XbAddr: targ = ?;
    XbadDa: NOT daWrite INSET use.da;
    XbSgRng: LENGTH(vec) ~= block.size;
    XbSgSz: block.size <= 0; $(IMPOSSIBLE, CARDINAL)
  ASSERTIONS
    PVMvmExists(pSeid);
  EFFECTS
    LET INTEGER relOffset = block.vloc.vAddr - use.vloc.vAddr
      IN 'SEGinstanceInfo(use.instance) =
           STRUCT(inst.gl, inst.growth, inst.refCount,
                  VECTOR(
                    FOR i FROM 1 TO LENGTH(inst.data)
                      : IF i INSET {relOffset + 1 .. relOffset + LENGTH(vec)}
                          THEN vec[i - relOffset]
                          ELSE inst.data[i]));

VFUN PVMretrieve(seid pSeid; pBlock block) -> VECTOR_OF INTEGER vec;
  $( retreives data from a mapped segment as specified by pBlock)
  DEFINITIONS
    segDes targ IS PVMblockToSeg(pSeid, block);
    useStruct use IS SEGuseInfo(pSeid, targ);
    instanceStruct inst IS SEGinstanceInfo(use.instance);
  EXCEPTIONS
    XbSgSz: block.size <= 0;          $(IMPOSSIBLE CARDINAL)
    XbSgDes: targ = ?;
  ASSERTIONS
    PVMvmExists(pSeid);
  DERIVATION
    VECTOR(FOR i FROM 1 TO block.size:
        inst.data[block.vloc.vAddr - use.vloc.vAddr + i - 1]);

OVFUN PVMbuild(seid pSeid; statusStruct st; modeStruct ms)
  -> STRUCT_OF(seid segSeid; segDes segd) result;               $(PVMbuild)
  $( builds a new segment with the specified parameters;
     an entry in the tii table i  also created for the segment;
     the results are the "" previously unused "" seid for the new segment and
     the "" previously unused "" segment designator; the newly created
     segment is mapped, indicating that it can be addressed immediately;
     discretionary access that the process allows itself to the segment, "m"
     must be a subset of the owner access specified in the tii information;
     "ms" of the new segment must be within the size limitations and fit into
     the mapped virtual memory space of the creating process)
  DEFINITIONS
    tiiStruct proTii IS TIIinfo(pSeid);
    tiiStruct segTii
      IS STRUCT(proTii.nd, ms, proTii.owner, proTii.group, proTii.priv);
    seid newSegSeid
      IS SENmakeSeid(exampleSegmentSeid,
                     SOME INTEGER i | NOT i INSET SEGinUseIndexSet());
    segDes sd IS SOME segDes sd1 | SEGuseInfo(pSeid, sd1) = ?;
  EXCEPTIONS
    XnPvStkSg: st.gl.sticky
                     AND NOT SMXhasPriv(pSeid, privStickySeg);
    XnPvLkSg: NOT st.gl.swappable AND NOT SMXhasPriv(pSeid,privLockSeg);
    XbadMode: ((daWrite INSET ms.ownerMode AND NOT
      daRead INSET ms.ownerMode) OR (daWrite INSET ms.groupMode
      AND NOT daRead INSET ms.groupMode) OR (daWrite INSET
      ms.allMode AND NOT daRead INSET ms.allMode));
    XbSgSz:NOT st.size INSET {1 .. MACmaxVaddr};
       $(test made only for 0;11/70,s card) $
    XvrtMcCfl:
      noHole(pSeid, st.size, st.vloc, st.growth, PVMmappedSegmentSet(pSeid));
    XpostEh: nSegs(pSeid) >= PVMmaxSegDes;
    RESOURCE_ERROR; $( ran out of table space or seid space)
  ASSERTIONS
    PVMvmExists(pSeid);
  EFFECTS
    'TIIinfo(newSegSeid) = segTii;
    'SEGinUseIndexSet()
      = SEGinUseIndexSet() UNION {SENseidIndex(newSegSeid)};
    'SEGinstanceInfo(newSegSeid) =
      STRUCT(st.gl, st.growth, 1,
             VECTOR(FOR i FROM 1 TO st.size : 0));
    'SEGuseInfo(pSeid, sd) = STRUCT(newSegSeid, st.vloc, ms.ownerMode);
    result = STRUCT(newSegSeid, sd);
    'PVMsegmentSet(pSeid) = PVMsegmentSet(pSeid) UNION {sd};
    'PVMmappedSegmentSet(pSeid)
     = PVMmappedSegmentSet(pSeid) UNION {sd};

OFUN PVMdestroy(seid pSeid; segDes segd);                     $(PVMdestroy)
  $(destroys the segment use indicated by pSeid and segd; if the segment is
    unsticky and otherwise unreferenced, the segment instance information is
    also deleted)
  DEFINITIONS
    useStruct use IS SEGuseInfo(pSeid, segd);
    seid segSeid IS use.instance;
    instanceStruct inst IS SEGinstanceInfo(segSeid);
  EXCEPTIONS
    KEsegNotHeld: NOT segd INSET PVMsegmentSet(pSeid);
  ASSERTIONS
  PVMvmExists(pSeid);
 EFFECTS
  'PVMsegmentSet(pSeid) = PVMsegmentSet(pSeid) DIFF {segd};
  'PVMmappedSegmentSet(pSeid) = PVMmappedSegmentSet(pSeid) DIFF {segd};
  'SEGuseInfo(pSeid, segd) = ?;
  IF (inst.refCount = 1 AND inst.gl.sticky = FALSE)
   THEN 'SEGinstanceInfo(segSeid) = ?
    AND 'SEGinUseIndexSet()
       = SEGinUseIndexSet() DIFF {SENseidIndex(segSeid)}
    AND 'TIIinfo(segSeid) = ?
   ELSE 'SEGinstanceInfo(segSeid) =
      STRUCT(inst.gl, inst.growth, inst.refCount - 1, inst.data);

$(*********************** high level storage allocation *******************)

OFUN PVMremap(seid pSeid; segDes in; virtualLocation vl; daType d;
       BOOLEAN vlFlg, daFlg; segDes out; INTEGER newSize;
       BOOLEAN nsFlg);                                   $(PVMremap)
 $( this function takes the currently mapped segment "out" and maps it out,
   while simultaneously mapping in the currently unmapped segment "in";
   this function can be used for mapping in "" without mapping out "" by
   letting "out" be the distinguished value SEGnullSeg; similarly, mapping
   out alone can be done by letting "in" be SEGnullSeg; a mapped in segment
   can have a new virtual location and a discretionary access specified
   optionally; a mapped out segment can have its size optionally changed;
   all these optional changes are specified by the values of BOOLEAN flags;
   the idSpace of the mapped in segment may not be changed; the mapped in
   segment must occupy a hole in the virtual memory)          .
 DEFINITIONS
  SET_OF segDes inSet IS IF in = SEGnullSeg THEN {} ELSE {in};
  SET_OF segDes outSet IS IF out = SEGnullSeg THEN {} ELSE {out};
  useStruct inUse IS SEGuseInfo(pSeid, in);
  instanceStruct inInst IS SEGinstanceInfo(inUse.instance);
  useStruct outUse IS SEGuseInfo(pSeid, out);
  seid outSeid IS outUse.instance;
  instanceStruct outInst IS SEGinstanceInfo(outSeid);
 EXCEPTIONS
  XbSgDes: NOT inSet SUBSET PVMsegmentSet(pSeid);
  XoutSgAldUmp: NOT outSet SUBSET PVMmappedSegmentSet(pSeid);
  XwOnlSg: daFlg AND daWrite INSET d AND NOT daRead INSET d;
  XinSgAldMap: {in} SUBSET PVMmappedSegmentSet(pSeid);
  XvrtMcCfl:
   in ~= SEGnullSeg
    AND noHole(pSeid, SEGsize(inUse.instance),
          IF vlFlg THEN vl ELSE inUse.vloc, inInst.growth,
          PVMmappedSegmentSet(pSeid) DIFF outSet);
  XbNewSz:
   nsFlg AND (NOT newSize INSET {1 .. MACmaxVaddr}
         OR outInst.gl.sharable
         OR NOT daWrite INSET outUse.da);
  RESOURCE_ERROR; $(for swapspace upon segment growth)
 ASSERTIONS
  PVMvmExists(pSeid);
 EFFECTS
  'PVMmappedSegmentSet(pSeid) = (PVMmappedSegmentSet(pSeid) DIFF outSet)
                     UNION inSet;
    'SEGuseInfo(pSeid, in)
      = STRUCT(inUse.instance, IF vlFlg THEN vl ELSE inUse.vloc,
              IF daFlg THEN d ELSE inUse.da);
   (nsFlg AND newSize ~= SEGsize(inUse.instance))
      => 'SEGinstanceInfo(outSeid)
           = STRUCT(outInst.gl, outInst.growth, outInst.refCount,
                    VECTOR(FOR i FROM 1 TO newSize
                           : IF i <= SEGsize(outSeid)
                                THEN outInst.data[i] ELSE 0));

$(*********************** segment sharing ********************************)

OVFUN PVMrendezvous(seid pSeid, segSeid; virtualLocation vl; daType d)
  *> segDes sd;                                         $(PVMrendezvous)
  $( creates a use for the segment named by segSeid; this segment appears
     inaccessible if the multilevel security model would consider it a
     violation of information flow)
  DEFINITIONS
    instanceStruct inst IS SEGinstanceInfo(segSeid);
    segDes segd IS SOME segDes sdl | SEGuseInfo(pSeid, sdl) = ?;
  EXCEPTIONS
    KEsegBadName:
      NOT (inst ~= ?
           AND (IF daWrite INSET d
                THEN SMXflow(pSeid, segSeid, {daRead, daWrite})
                ELSE SMXflow(pSeid, segSeid, {daRead})));
    XbadDa: NOT SMXdap(pSeid, segSeid, d);
    XdupSg: EXISTS segDes sdl
              : SEGuseInfo(pSeid, sdl).instance = segSeid;
    XwOnlSg: daWrite INSET d AND NOT daRead INSET d;
    XvrtMmCfl: noHole(pSeid, SEGsize(segSeid), vl, inst.growth,
                      PVMmappedSegmentSet(pSeid));
    XpostEh: nSegs(pSeid) >= PVMmaxSegDes;
    XnotSh:  NOT inst.gl.sharable;
  ASSERTIONS
    PVMvmExists(pSeid);
  EFFECTS
    'SEGuseInfo(pSeid, segd) = STRUCT(segSeid, vl, d);
    'SEGinstanceInfo(segSeid)
      = STRUCT(inst.gl,inst.growth, inst.refCount + 1, inst.data);
    'PVMsegmentSet(pSeid) = PVMsegmentSet(pSeid) UNION {segd};
    'PVMmappedSegmentSet(pSeid)
      = PVMmappedSegmentSet(pSeid) UNION {segd};
    sd = segd;

OFUN PVMcopySeg(seid fromSeid, toSeid; segDes sd);            $(PVMcopySeg)
  $( copies a segment from the virtual memory "fromSeid" to the virtual
     memory "toSeid"; both virtual memories must exist; the segment
     designator sd must exist in "fromSeid" but not in "toSeid"; used by
     the module that sets up virtual memories for new processes)
  DEFINITIONS
    useStruct use IS SEGuseInfo(fromSeid, sd);
    seid oldSeid IS use.instance;   instanceStruct inst IS SEGinstanceInfo(oldSeid);
    seid newSeid
      IS SENmakeSeid(exampleSegmentSeid,
                     SOME INTEGER i | NOT i INSET SEGinUseIndexSet());
    tiiStruct stii IS TIIinfo(oldSeid);
    tiiStruct ptii IS TIIinfo(toSeid);
  ASSERTIONS
    PVMvmExists(fromSeid);
    PVMvmExists(toSeid);
    sd INSET PVMsegmentSet(fromSeid);
    NOT sd INSET PVMsegmentSet(toSeid);
  EFFECTS
    'SEGinstanceInfo(newSeid) = STRUCT(inst.gl, inst.growth, 1, inst.data);
    'SEGuseInfo(toSeid, sd) = STRUCT(newSeid, use.vloc, use.da);
    'TIIinfo(newSeid) = STRUCT(stii.nd, stii.ms, ptii.owner, ptii.group,
                               stii.priv);
    'PVMsegmentSet(toSeid) = PVMsegmentSet(toSeid) UNION {sd};
    sd INSET PVMmappedSegmentSet(fromSeid) =>
    'PVMmappedSegmentSet(toSeid) = PVMmappedSegmentSet(toSeid) UNION {sd};
    'SEGinUseIndexSet()
    = SEGinUseIndexSet() UNION {SENseidIndex(newSeid)};

$(********************** segment status manipulation ***********************)

VFUN PVMgetSegmentStatus(seid pSeid, segSeid; segDes segd) *> statusStruct ss;
                                                       $(PVMgetSegmentStatus)
  $( returns the status information ** which is much of the global
     information ** for the segment; the segment must exist in the segment
     set of the requesting process)
DEFINITIONS
          seid checkSeid IS IF segd=SEGnullSeg THEN segSeid ELSE
                                SEGuseInfo(pSeid,segd).instance;
          segDes desrefl  IS SOME segDes s|
                          SEGuseInfo(pSeid,s).instance=checkSeid;
          segDes desref   IS IF desrefl = ? THEN SEGnullSeg
                                ELSE desrefl;
  EXCEPTIONS
    XbSgSd: IF segd = SEGnullSeg THEN  (SEGinstanceInfo(segSeid) = ?
                     OR NOT SMXflow(pSeid, segSeid, {daRead}))
            ELSE NOT segd INSET PVMsegmentSet(pSeid);
  ASSERTIONS
    PVMvmExists(pSeid);
  DERIVATION
    STRUCT(     SEGinstanceInfo(checkSeid).gl,
                SEGinstanceInfo(checkSeid).growth,
                SEGsize(checkSeid),
                checkSeid,
                desref,
                IF desref=SEGnullSeg THEN STRUCT(nullDomain,nullSpace,0)
                    ELSE SEGuseInfo(pSeid,desref).vloc,
                IF desref=SEGnullSeg THEN {}
                    ELSE SEGuseInfo(pSeid,desref).da,
                IF desref=SEGnullSeg THEN FALSE
                    ELSE IF desref INSET PVMmappedSegmentSet(pSeid) THEN TRUE
                    ELSE FALSE
        );

OFUN PVMsetSegmentStatus(seid pSeid, segSeid; globalData glo);
                                                       $(PVMsetSegmentStatus)
  $( changes the status information for the segment; certain privileges
     are required;)
  DEFINITIONS
    instanceStruct i IS SEGinstanceInfo(segSeid);
  EXCEPTIONS
  XbSgSd: NOT (i ~= ? AND SMXflow(pSeid, segSeid, {daRead, daWrite}));
  Xsharable: i.gl.sharable;
    XswapEh:
      glo.swappable AND NOT i.gl.swappable
        AND NOT SMXhasPriv(pSeid, privLockSeg);
    XnPvStkSg:
      glo.sticky AND NOT i.gl.sticky
        AND NOT SMXhasPriv(pSeid, privStickySeg);
  ASSERTIONS
    PVMvmExists(pSeid);
  EFFECTS
    'SEGinstanceInfo(segSeid) = STRUCT(glo, i.growth, i.refCount, i.data);

$(************************ level getting and setting ************************)

VFUN PVMgetSegmentLevel(seid pSeid, segSeid) *> tiiStruct tii;
                                                         $(PVMgetSegmentLevel)
  EXCEPTIONS
    $(LR ** these must be finalized)
    XbSgSd: NOT (SEGinstanceInfo(segSeid) ~= ?
                    AND SMXflow(pSeid, segSeid, {daRead}));
    XbadDa: NOT SMXdap(pSeid, segSeid, {daRead});
  DERIVATION
    TIIinfo(segSeid);

OFUN PVMsetSegmentLevel(seid pSeid, segSeid; tiiStruct ntii);
                                                         $(PVMsetSegmentLevel)
  DEFINITIONS
    tiiStruct otii IS TIIinfo(segSeid);
  EXCEPTIONS
    XbSgSd: NOT (SEGinstanceInfo(segSeid) ~= ?
                      AND SMXflow(pSeid, segSeid, {daRead, daWrite}));
    XbadDa: NOT SMXdap(pSeid, segSeid, {daWrite});
    XshSg: SEGinstanceInfo(segSeid).gl.sharable;
    XnoSLev: otii.nd ~= ntii.nd
                AND NOT SMXhasPriv(pSeid, privLockSeg);
    XwOnlSg: otii.ms ~= ntii.ms
       AND ((daWrite INSET ntii.ms.ownerMode
                 AND NOT daRead INSET ntii.ms.ownerMode)
              OR (daWrite INSET ntii.ms.groupMode
                 AND NOT daRead INSET ntii.ms.groupMode)
              OR (daWrite INSET ntii.ms.allMode
                 AND NOT daRead INSET ntii.ms.allMode));
    XchgSgOwn:
      (otii.ms ~= ntii.ms OR otii.owner ~= ntii.owner
          OR otii.group ~= ntii.group)
        AND otii.owner ~= TIIinfo(pSeid).owner;
    XnoSPriv: otii.priv ~= ntii.priv
                       AND NOT SMXhasPriv(pSeid, privModifyPriv);
  EFFECTS
    'TIIinfo(segSeid) = ntii;

```

```

END_MODULE
```

```
  $("    MODULE:          pvp.specs (version 4.12)
          CONTENTS:        Virtual Memory ** Process Supp
          TYPE:
          LAST CHANGED:    12/23/80, 16:59:00
   ")


MODULE pvp

$( this module contains operations that support the process module's use of
   the virtual memory mechanism.  This is entirely a procedure abstraction.
   It is specified in terms of V*functions of other modules, but is
   implemented by a program in terms of operations of other modules.
   This sequence will be included in the comments for each of the operations.
   This module has no state of its own except for parameters for immediate
   segments.)

     TYPES

     $(from mac)
vAddrType: {0 .. MACmaxVAddr};

     $(from smx)
nonDisType: STRUCT_OF(
               INTEGER securityLevel; SET_OF securityCat securityCatS;
               INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
daType: SET_OF daMode;
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
tiiStruct: STRUCT_OF(nonDisType nd;
               modeStruct ms; INTEGER owner, group; SET_OF privType priv);

     $(from pvm)
virtualLocation:
  STRUCT_OF(domainType domain; spaceType idSpace; vAddrType vAddr);
globalData:
  STRUCT_OF(BOOLEAN sharable, swappable, sticky, memAdvise, executable;
              direction growth);
instanceStruct:
  STRUCT_OF(globalData gl; direction growth; INTEGER refCount; VECTOR_OF INTEGER data);
useStruct: STRUCT_OF(seid instance; virtualLocation vloc; daType da);


     PARAMETERS

virtualLocation PVMimmVloc; $(location for immediate segment)
segDes PVMimmDes; $(designator for immediate text segment)
segDes PVMargDes; $(designator for argument segment)
virtualLocation PVMargVloc;  $(location for argument segment)


     DEFINITIONS

     $(from seg)
INTEGER SEGsize(seid segSeid) IS LENGTH(SEGinstanceInfo(segSeid).data);

     $(from pvm)
```

```
INTEGER nSegs(seid pSeid)
  IS CARDINALITY({segDes sd | SEGuseInfo(pSeid, sd) ~= ?});

BOOLEAN PVMvmExists(seid pSeid) IS PVMsegmentSet(pSeid) ~= ?;

SET_OF INTEGER addrRegRange(INTEGER vAddr, size; direction d) IS
  IF d  = up
    THEN {vAddr / MACmaxOffset .. (vAddr + size - 1) / MACmaxOffset}
    ELSE {(vAddr - size + 1) / MACmaxOffset .. vAddr / MACmaxOffset};
  $( gives the range of address registers used by a segment as a function of
     its start address, size, and growth direction)

SET_OF INTEGER addrRegRangeSeg(seid pSeid; segDes s) IS
  LET useStruct use = SEGuseInfo(pSeid, s)
    IN addrRegRange(use.vloc.vAddr, SEGsize(use.instance),
                    SEGinstanceInfo(use.instance).gl.growth);
  $( gives the range of address registers used by a segment as a function of
     the process id and the segment designator)

BOOLEAN noHole(seid pSeid; INTEGER size; virtualLocation vl; direction d;
               SET_OF segDes ssd) IS
  NOT addrRegRange(vl.vAddr, size, d) SUBSET {0 .. MACmaxReg}
    OR (EXISTS segDes s | s INSET ssd; useStruct use = SEGuseInfo(pSeid, s)
          : use.vloc.idSpace = vl.idSpace
              AND use.vloc.domain = vl.domain
              AND addrRegRangeSeg(pSeid, s)
                      INTER addrRegRange(vl.vAddr, size, d) ~= {});
  $(TRUE if a segment described by size, vl, and direction will NOT fit into
    a hole in the address space designated by pSeid and ssd; this includes
    testing for virtual memory underflow and overflow)

    EXTERNALREFS

    FROM mac:
INTEGER MACmaxVAddr, MACmaxOffset, MACmaxReg;

    FROM smx:
seid: DESIGNATOR;
        privType: {
                                privFileUpdateStatus,  privLink,
                                privLockSeg,           privModifyPriv,
                                privMount,             privSetLevel,
                                privStickySeg,         privSetPath,
                                privViolSimpSecurity,  privViolStarSecurity,
                                privViolSimpIntegrity, privViolStarIntegrity,
                                privViolDiscrAccess,   privRealizeExecPermission,
                                privSignal,            privWalkPTable,
                                privHalt,              privKernelCall,
                                privViolCompartments,  privSetComm,
                                privImmigrate
                                };
daMode: {daRead, daWrite, daExecute};
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;
domainType: {nullDomain, kernelDomain,supervisorDomain,userDomain};
VFUN SENseidNsp(seid s) -> INTEGER nsp;
VFUN TIIinfo(seid anySeid) *> tiiStruct tiiSt;
VFUN SMXflow(seid pSeid, oSeid; daType d) *> BOOLEAN b;
VFUN SMXdap(seid pSeid, oSeid; daType d) *> BOOLEAN b;
VFUN SENseidIndex(seid anySeid) *> INTEGER index;

    FROM pvm:
segDes: DESIGNATOR;
spaceType: {nullSpace,iSpace, dSpace};
direction: {up, down};
seid exampleSegmentSeid; $(used for segment creation)
INTEGER PVMmaxSegDes;
VFUN SEGinstanceInfo(seid segSeid) *> instanceStruct is;
VFUN SEGuseInfo(seid pSeid; segDes segd) *> useStruct us;
VFUN PVMsegmentSet(seid pSeid) *> SET_OF segDes segSet;
VFUN PVMmappedSegmentSet(seid pSeid) *> SET_OF segDes mappedSet;
VFUN SEGinUseIndexSet() *> SET_OF INTEGER si;


    ASSERTIONS

PVMimmVloc.domain = supervisorDomain;
PVMimmVloc.idSpace = iSpace;
PVMargVloc.domain = supervisorDomain;
PVMargVloc.idSpace = dSpace;
  $( constraints on parameters)


    FUNCTIONS

$(************************* support for PSTreleaseProcess ****************)

OFUN PVPreleaseProcessSupport(seid pSeid);
  $( This function supports PROreleaseProcess by deleting all segments in
     the virtual memory named by "pSeid" and then deleting the virtual memory
     itself)
  DEFINITIONS
    seid segSeid(segDes sd) IS SEGuseInfo(pSeid, sd).instance;
    instanceStruct inst(segDes sd) IS SEGinstanceInfo(segSeid(sd));
  ASSERTIONS
    PVMvmExists(pSeid);
  EFFECTS
    FORALL segDes sd | SEGuseInfo(pSeid, sd) ~= ?
      : 'SEGuseInfo(pSeid, sd) = ?
        AND (IF inst(sd).refCount = 1 AND inst(sd).gl.sticky = FALSE
               THEN 'SEGinstanceInfo(segSeid(sd)) = ?
                    AND 'TIIinfo(segSeid(sd)) = ?
               ELSE
                'SEGinstanceInfo(segSeid(sd))
                 = STRUCT(inst(sd).gl, inst(sd).growth,
                       inst(sd).refCount*1, inst(sd).data))·
    'PVMsegmentSet(pSeid) = ?;
    'PVMmappedSegmentSet(pSeid) = ?;


$( NOTE ** There are two special segments that are appropriate to the invoke
   and spawn operations:  the argument segment "arg", already in the virtual
```

```
    memory, which contains the data to be used by the initialized process;
    and the immediate segment, "immSeid", which contains the code for the
    process ** this code is also called the process bootstrapper)

$(************************ support for PSTinvoke ************************)

OFUN PVPinvokeSupport(seid pSeid, immSeid; segDes arg);   $(PVPinvokeSupport)
  $( This function sets up the virtual memory of  a  process  for
     invocation; the new mapped set contains all previously mapped supervisor
     segments and the argument and immediate segments; the argument segment
     is mapped to a different virtual location, and a use is created for the
     the immediate segment)
  DEFINITIONS
    instanceStruct immInst IS SEGinstanceInfo(immSeid);
    useStruct argUse IS SEGuseInfo(pSeid, arg);
    seid argSeid IS argUse.instance;
    instanceStruct argInst IS SEGinstanceInfo(argSeid);
  EXCEPTIONS
    XbSgDes: argUse = ?;
    XshrSg: argInst.gl.sharable;
    XbSgSd: NOT (immInst ~= ?
                          AND SMXflow(pSeid, immSeid, {daRead}));
    XsgNoAcc: NOT SMXdap(pSeid, immSeid, {daExecute});
    XbSgRng:
      NOT addrRegRange(PVMargVloc.vAddr, SEGsize(argSeid), argInst.gl.growth)
        SUBSET {0 .. MACmaxReg};
    XbSgRng:
      NOT addrRegRange(PVMimmVloc.vAddr, SEGsize(immSeid), immInst.gl.growth)
        SUBSET {0 .. MACmaxReg};
  XpostEh:
    nSegs(pSeid) >= PVMmaxSegDes;
  ASSERTIONS
    PVMvmExists(pSeid);
  EFFECTS
    'SEGuseInfo(pSeid, PVMargDes)
      = STRUCT(argUse.instance, PVMargVloc, argUse.da);
    $( create a reference to the immediate segment)
    'SEGuseInfo(pSeid, PVMimmDes)
       = STRUCT(immSeid, PVMimmVloc, {daRead, daExecute});
    'SEGinstanceInfo(immSeid)
       = STRUCT(immInst.gl, immInst.growth,
                immInst.refCount + 1, immInst.data);
    $( add the immediate segment to the address space)
    'PVMsegmentSet(pSeid) = PVMsegmentSet(pSeid) UNION {PVMimmDes,PVMargDes};
    $( unmap all segments except the argument segment,
       and the immediate segment)
    'PVMmappedSegmentSet(pSeid)
      = {PVMargDes, PVMimmDes};
    $( remap the argument segment)

$(**********************support for PSTspawn ****************************)

OFUN PVPspawnSupport(seid parent, child; seid immSeid; segDes arg);
                                                           $(PVPspawnSupport)
  $(creates a new address space named by child and inserts into it two
    segment uses ** the argument segment, arg, which is copied from the parent
  address space, but occupies a different position '' PVMargDes ''
  and the immediate segment, immSeid, which is shared)
 DEFINITIONS
  instanceStruct immInst IS SEGinstanceInfo(immSeid);
  useStruct argUse IS SEGuseInfo(parent, arg);
  seid argSeid IS argUse.instance;
  instanceStruct argInst IS SEGinstanceInfo(argSeid);
  seid argCopy
   IS SOME seid s | SENseidNsp(s) = SENseidNsp(exampleSegmentSeid)
                    AND SEGinstanceInfo(s) = ?;
  tiiStruct aTii IS TIIinfo(argSeid);
  tiiStruct pTii IS TIIinfo(parent);
  tiiStruct nTii IS STRUCT(aTii.nd, aTii.ms, pTii.owner, pTii.group,
                      aTii.priv);
 EXCEPTIONS
  XbSgSd: NOT (immInst ~= ? AND SMXflow(parent,immSeid,{daRead}));
  XsgNoAcc: NOT SMXdap(parent, immSeid, {daExecute});
  XshrSg: argInst.gl.sharable;
  XnWrtArgSg: NOT daWrite INSET argUse.da;
  XbSgRng:
   NOT addrRegRange(PVMargVloc.vAddr, SEGsize(argSeid), argInst.gl.growth)
    SUBSET {0 .. MACmaxReg};
  XbSgRng:
   NOT addrRegRange(PVMimmVloc.vAddr, SEGsize(immSeid), immInst.gl.growth)
    SUBSET {0 .. MACmaxReg};
  RESOURCE_ERROR;
 ASSERTIONS
  PVMvmExists(parent);
  NOT PVMvmExists(child);
 EFFECTS
  $( create a copy of the argument segment)
  'TIIinfo(argCopy) = nTii;
  'SEGinstanceInfo(argCopy) = STRUCT(argInst.gl, argInst.growth,1,
                           argInst.data);
  'SEGuseInfo(child, PVMargDes) = STRUCT(argCopy, argUse.vloc, argUse.da);
  $( create a use for immSeid in child)
  'SEGuseInfo(child, PVMimmDes)
    = STRUCT(immSeid, PVMimmVloc, {daRead, daExecute});
  'SEGinstanceInfo(immSeid)
    = STRUCT(immInst.gl,immInst.growth,immInst.refCount + 1, immInst.data);
  'PVMsegmentSet(child) = {PVMimmDes, PVMargDes};
  'PVMmappedSegmentSet(child) = {PVMimmDes, PVMargDes};

$(******************** support for PROfork ********************************)

OFUN PVPforkSupport(seid parent, child);                    $(PVPforkSupport)
 $(creates a new virtual memory, child, that is a copy of parent; some
  segments are copied and others are merely shared; if a segment is
  sharable in the parent process, it is not copied, but a use corresponding
  to the instance in parent is created instead; if the segment is not
  sharable, then a new instance of the segment is created, requiring the
  allocation of an unused seid; in either case, corresponding segments have
  identical segment designators in both processes; much mechanism in this
  specification is devoted to describing the set of new seids created and
  the mapping of this set onto the set of new segment instances)
 DEFINITIONS
```

```
    INTEGER nCopies
      IS CARDINALITY
           ({segDes sd | SEGinstanceInfo(SEGuseInfo(parent,
                                                    sd).instance).gl.sharable
                          = FALSE});
      $(number of nonsharable segments in parent process)
    SET_OF INTEGER indexCopySet
      IS SOME SET_OF INTEGER si
           | CARDINALITY(si) = nCopies AND si INTER SEGinUseIndexSet() = {};
    SET_OF seid copySet
      IS {seid s | SENseidNsp(s) = SENseidNsp(exampleSegmentSeid)
            AND SENseidIndex(s) INSET indexCopySet};
      $(actual set of new seids)
    useStruct use(segDes segd) IS SEGuseInfo(parent, segd);
    instanceStruct inst(segDes segd)
      IS SEGinstanceInfo(use(segd).instance);
  EXCEPTIONS
    RESOURCE_ERROR;
  ASSERTIONS
    PVMvmExists(parent);
    NOT PVMvmExists(child);
  EFFECTS
    'PVMsegmentSet(child) = PVMsegmentSet(parent);
    'PVMmappedSegmentSet(child) = PVMmappedSegmentSet(parent);
    FORALL segDes segd | SEGuseInfo(parent, segd).instance ~= ?
      : (IF inst(segd).gl.sharable
           THEN
             'SEGuseInfo(child, segd) = use(segd)
             AND 'SEGinstanceInfo(use(segd).instance) =
               STRUCT(inst(segd).gl, inst(segd).growth,
                      inst(segd).refCount + 1, inst(segd).data)
           ELSE
             (LET seid copy INSET copySet
               IN 'TIIinfo(copy) = TIIinfo(use(segd).instance)
                  AND 'SEGinstanceInfo(copy) =
                        STRUCT(inst(segd).gl, inst(segd).growth,
                               1, inst(segd).data)
                  AND 'SEGuseInfo(child, segd) =
                        STRUCT(copy, use(segd).vloc, use(segd).da)
                  AND (FORALL segDes segd1 ~= segd
                        : 'SEGuseInfo(child, segd1).instance
                            ~= 'SEGuseInfo(child, segd).instance));
             'SEGinUseIndexSet() = SEGinUseIndexSet()UNION indexCopySet;


END_MODULE
```

```
$("      MODULE:          smx.specs (version 4.14)
         CONTENTS:        Security Model
         TYPE:            SPECIAL.specifications
         LAST CHANGED:    12/23/80, 17:40:14
  ")


MODULE smx


$( This module now includes what used to be the contents of smx, prv, tii,
   syl, and sen modules)

    TYPES

    $(from smx ** exportable)
seid: DESIGNATOR;
secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
                   tExtent, tNull};
        privType: {
                          privFileUpdateStatus,  privLink,
                          privLockSeg,           privModifyPriv,
                          privMount,             privSetLevel,
                          privStickySeg,         privSetPath,
                          privViolSimpSecurity,  privViolStarSecurity,
                          privViolSimpIntegrity, privViolStarIntegrity,
                          privViolDiscrAccess,   privRealizeExecPermission,
                          privSignal,            privWalkPTable,
                          privHalt,              privKernelCall,
                          privViolCompartments,  privSetComm,
                          privImmigrate
                          };

daMode: {daRead, daWrite, daExecute};
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;
domainType: {nullDomain, kernelDomain,supervisorDomain,userDomain};

    $(from smx ** redeclarable)
nonDisType: STRUCT_OF(
               INTEGER securityLevel; SET_OF securityCat securityCatS;
               INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
  $(integrityCat is typically the null set)
daType: SET_OF daMode;
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
tiiStruct: STRUCT_OF(nonDisType nd;
               modeStruct ms; INTEGER owner, group; SET_OF privType priv);


    PARAMETERS

INTEGER SENmaxIndex $(maximum index component of a seid, 2^24 - 1),
        SENmaxNsp $(maximum nsp component of a seid, 2^8 - 1);
INTEGER SENlowLevel, $(system low level)
        SENhighLevel; $(system high level)


    ASSERTIONS

FORALL seid s1; seid s2: (s1 = s2) = (SENseidNsp(s1) = SENseidNsp(s2)
                                  AND SENseidIndex(s1) = SENseidIndex(s2));
   $(this states that the nsp and index are isomorphic with the seid, and
     thus uniquely identify it)

SENlowLevel < SENhighLevel;
   $(defines the "greater than" relation for security in terms of the integer
     relation ">")

SYLgetHigh() INSET {SENlowLevel .. SENhighLevel};
    $(the current system high level is always within range)

FORALL seid s | TIIinfo(s) ~= ?
  : LET nonDisType ndl = TIIinfo(s).nd
      IN ndl.securityLevel INSET {SENlowLevel .. SENhighLevel}
           AND ndl.integrityLevel INSET {SENlowLevel .. SENhighLevel};
   $(restricts the values of the security and integrity levels for any
     existing objects)


    FUNCTIONS

$(******************** sen ** state functions ******************************)

VFUN SENseidNsp(seid anySeid) -> INTEGER nsp;                    $(SENseidNsp)
  $( this is the nsp table entry component of a seid)
  INITIALLY
    nsp INSET {0 .. SENmaxNsp}; $(constrained by assertion above)

VFUN SENseidIndex(seid anySeid) -> INTEGER index;              $(SENseidIndex)
  $( this is the index component for a seid)
  INITIALLY
    index INSET {0 .. SENmaxIndex};
   $(also characterized by assertion)

VFUN SENnspType(INTEGER nsp) -> secureEntityType set;            $(SENnspType)
  $( gives the type information as a function of the nsp component of
     a seid)
  INITIALLY NOT nsp INSET {0 .. SENmaxNsp} => set = ?;

$(****************** sen ** seid and nsp manipulation *********************)

VFUN SENseidToInt(seid anySeid) -> INTEGER i;                  $(SENseidToInt)
  $(gives the integer corresponding to a given seid)
  DERIVATION
    SENseidIndex(anySeid) + 2^24 * SENseidNsp(anySeid);

VFUN SENseidType(seid s) -> secureEntityType set;               $(SENseidType)
  $( returns the type information pertaining to a given seid)
  DERIVATION
    LET secureEntityType set1 = SENnspType(SENseidNsp(s))
      IN IF set1 = ? THEN tNull ELSE set1;

```

```
VFUN SENmakeSeid(seid exampleSeid; INTEGER index) -> seid rSeid;
                                                            $(SENmakeSeid)
  $( forms a seid with an nsp the same as the example seid and the given
     index; seid allocation is now done by the type managers for the
     objects in question, allowing seids to be reused in the case of
     objects that are dynamically allocated)
  ASSERTIONS
    index INSET {0 .. SENmaxIndex};
  DERIVATION
    SOME seid s | SENseidNsp(s) = SENseidNsp(exampleSeid)
                    AND SENseidIndex(s) = index;

$(-------------------- syl functions ---------------------------------------)

VFUN SYLgetHigh() -> INTEGER level;                         $(SYLgetHigh)
  $( represents the current highest security level for the system)
  HIDDEN;
  INITIALLY
    level = SENhighLevel;

OFUN SYLsetHigh(INTEGER level);                             $(SYLsetHigh)
  $( sets the current highest security level for the system to the specified
     value)
  EXCEPTIONS
    XbSysLev: NOT level INSET {SENlowLevel .. SENhighLevel};
  EFFECTS
    'SYLgetHigh() = level;

$(------------------ tii state function ------------------------------------)

VFUN TIIinfo(seid s) -> tiiStruct st;                          $(TIIinfo)
  $(returns the type-independent information for a system object; this
    information includes discretionary, non-discretionary, and domain access
    controls, privileges, and the owner and group for the object)
  HIDDEN;
  INITIALLY  st = ?;

$(-------------------- smx functions ---------------------------------------)

VFUN SMXhasPriv(seid pSeid; privType privl) -> BOOLEAN b;       $(SMXhasPriv)
  $( tells whether a given object -- usually a process -- has a particular
     privilege)
  DERIVATION
    IF TIIinfo(pSeid) ~= ? THEN privl INSET TIIinfo(pSeid).priv ELSE FALSE;

VFUN SMXcompare(seid lSeid, hSeid) -> BOOLEAN b;            $(SMXcompare)
  $( Primitive comparison telling whether lSeid <= hSeid.  No privileges
     are accounted for.)
  DEFINITIONS
    tiiStruct lTii IS TIIinfo(lSeid);
    tiiStruct hTii IS TIIinfo(hSeid);
  DERIVATION
    IF lTii ~= ? AND hTii ~= ?
      THEN     lTii.nd.securityLevel <= hTii.nd.securityLevel
           AND lTii.nd.securityCatS SUBSET hTii.nd.securityCatS
           AND lTii.nd.integrityLevel >= hTii.nd.integrityLevel
           AND hTii.nd.integrityCatS SUBSET lTii.nd.integrityCatS
      ELSE FALSE;

VFUN SMXcompWpriv(seid lSeid,hSeid; BOOLEAN privl,priv2) =>BOOLEAN b;
                                                        $(SMXcompWpriv)
  $(Primitive comparison telling whether lSeid <= hSeid, but
    with privileges taken into account.  See SMXflow comments re.
    need for both SMXcompare and SMXcompWpriv.)
  DEFINITIONS
    tiiStruct lTii IS TIIinfo(lSeid);
    tiiStruct hTii IS TIIinfo(hSeid);
  DERIVATION
    IF lTii ~= ? AND hTii ~= ?
      THEN  (privl
            OR
             (    lTii.nd.securityLevel <= hTii.nd.securityLevel
             AND lTii.nd.securityCatS SUBSET hTii.nd.securityCatS
             )
            )

            AND

            (priv2
            OR
             (    lTii.nd.integrityLevel >= hTii.nd.integrityLevel
              AND hTii.nd.integrityCatS SUBSET lTii.nd.integrityCatS
             )
            )
      ELSE FALSE;

VFUN SMXflow(seid pSeid, oSeid; daType d) => BOOLEAN b;           $(SMXflow)
  $( tells whether a given subject "pSeid" can access a given object "oSeid"
     with the information flow specified by "flow", according to the
     constraints of the military multilevel security model; these constraints
     do not apply if the subject has the proper privilege)
   $("  The reason for having both SMXcompare and SMXcompWpriv is that
     the MLS tools have axiomatized SMXcompare, i.e., do not account
     for privileges;  the implementation does account for privileges.
     The following relationships are true:
        FORALL seid s1,s2|TIIinfo(s1)~=? AND TIIinfo(s2)~=? :
           SMXcompWpriv(s1,s2,TRUE,TRUE) = TRUE
       AND SMXcompWpriv (s1,s2,FALSE,FALSE) = SMXcompare(s1,s2)
       AND SMXcompare(s1,s2) => SMXcompWpriv(s1,s2,TRUE,FALSE)
       AND SMXcompare(s1,s2) => SMXcompWpriv(s1,s2,FALSE,TRUE);
   ")
  DERIVATION
        (daWrite INSET i =>
           SMXcompare(pSeid, oSeid)
           OR
           SMXcompWpriv(pSeid,oSeid,SMXhasPriv(pSeid,privViolStarSecurity),
                        SMXhasPriv(pSeid,privViolSimpIntegrity))
        )
    AND
        (daRead INSET d =>
           SMXcompare(oSeid, pSeid)
           OR
           SMXcompWpriv(oSeid,pSeid,SMXhasPriv(pSeid,privViolSimpIntegrity),
                        SMXhasPriv(pSeid,privViolStarIntegrity))
        )
        ;

VFUN SMXdap(seid pSeid, oSeid; daType d) *> BOOLEAN b;             $(SMXdap)
  $( tells whether a given subject "pSeid" can access a particular object
     "oSeid" according to the discretionary access rules of the system **
     similar to those of UNIX)
  DEFINITIONS
    tiiStruct p IS TIIinfo(pSeid);
    modeStruct mst IS TIIinfo(oSeid).ms;
    tiiStruct o IS TIIinfo(oSeid);
    BOOLEAN access(daType requested, allowed)
      IS requested
           SUBSET allowed
                    UNION (IF SMXhasPriv(pSeid, privRealizeExecPermission)
                                AND daExecute INSET allowed
                             THEN {daRead}
                             ELSE {});
  DERIVATION
    IF o ~= ?
      THEN SMXhasPriv(pSeid, privViolDiscrAccess)
             OR access(d, mst.allMode)
             OR (p.group = o.group AND access(d, mst.groupMode))
             OR (p.owner = o.owner AND access(d, mst.ownerMode))
      ELSE FALSE;

$(*************** tii ** extraction and insertion functions***************)

OFUN TIIcreateEntityLevel(seid oSeid; tiiStruct ntii);$(TIIcreateEntityLevel)
  $( Initializes the tii information for an object "oSeid"; the object must
     not have currently defined tii information; this is a service function
     required for the creation of all types of objects in KSOS)
  ASSERTIONS
    TIIinfo(oSeid) = ?;
    ntii.nd.securityLevel INSET {SENlowLevel .. SYLgetHigh()};
    ntii.nd.integrityLevel INSET {SENlowLevel .. SENhighLevel};
  EFFECTS
    'TIIinfo(oSeid) = ntii;

VFUN TIIgetEntityLevel(seid pSeid, oSeid) *> tiiStruct ntii;
                                                         $(TIIgetEntityLevel)
  $(Retrieves the tii information of an object named by "oSeid", as
    directed by process "pSeid"; mandatory and discretionary checks are also
    performed;  this function is used by
    functions of the object*maintaining modules, which provide status
    information to the object that is concerned with getting and setting
    object levels)
  EXCEPTIONS
    XnoObj: NOT(TIIinfo(oSeid) ~= ?
                    AND SMXflow(pSeid, oSeid, {daRead}));
  DERIVATION
    TIIinfo(oSeid);

OFUN TIIsetEntityLevel(seid pSeid, oSeid; tiiStruct ntii);
```

```
                                                      $(TIIsetEntityLevel)
  $( sets the type-independent information for an existing object
     "oSeid"  to  the  new  value  "ntii", as directed by process
     "pSeid"; the privilege to set level is always required; if an
     object's privileges are to be modified, the  privilege to modify
     privileges is required; because only privileged programs are allowed to
     invoke this function, no other security checks -- either mandatory
     or discretionary -- are made)
  DEFINITIONS
    tiiStruct otii IS TIIinfo(oSeid);
    BOOLEAN sameOwner
      IS TIIinfo(pSeid).owner = TIIinfo(oSeid).owner
           OR SMXhasPriv(pSeid, privSetLevel);
  EXCEPTIONS
    $(privilege checking occurs in lev module)
    XnSPriv:
      otii.priv ~= ntii.priv AND NOT SMXhasPriv(pSeid, privModifyPriv);
    XnoObj: otii = ?
       OR (otii.ms ~= ntii.ms AND NOT SMXflow (pSeid,oSeid, {daWrite})
             AND NOT sameOwner);
     KEtiiNoSetLev:
      otii.nd ~= ntii.nd AND NOT SMXhasPriv(pSeid,privSetLevel);
     KEfiiNoSetPriv:
       otii.priv ~= ntii.priv AND NOT SMXhasPriv(pSeid,privSetLevel);
     KEtiiNoSetOwner:
       otii.owner ~= ntii.owner AND NOT SMXhasPriv(pSeid,privSetLevel) AND (pSeid ~= oSeid,
     KEtiiNoSetGroup:
       otii.group ~= ntii.group AND NOT SMXhasPriv(pSeid,privSetLevel) AND (pSeid ~= oSeid


  ASSERTIONS
    ntii.nd.securityLevel INSET {SENlowLevel .. SYLgetHigh()};
    ntii.nd.integrityLevel INSET {SENlowLevel .. SENhighLevel};
  EFFECTS
    'TIIinfo(oSeid) = ntii;

OFUN TIIclearEntityLevel(seid oSeid);                 $(TIIclearEntityLevel)
  $( deletes the type-independent information for an object "oSeid"; this
     function is used for implementing functions that delete objects -- and
     thus must also delete the tii info)
  EFFECTS
    'TIIinfo(oSeid) = ?;


END_MODULE
```

```
$("     MODULE:          spf.specs (version 4.6)
        CONTENTS:        Special Functions
        TYPE:            SPECIAL.specifications
        LAST CHANGED:    7/28/80, 09:56:25
  ")


MODULE spf


$( This module will contain the specifications for functions which do not
   properly fit into any of the other kernel modules.  Functions so envisioned
   will synchronize the file system, cause the system to halt, and set the
   maximum system security level.  As of now these functions are left
   unspecified)


    TYPES

SPFfunctionType: {syncSPF, immSegSPF, sysHaltSPF, levelSetSPF};
SPFargs: VECTOR_OF INTEGER;


    EXTERNALREFS

    FROM smx:
seid: DESIGNATOR;


    FUNCTIONS

OFUN SPFspecialFunction(seid pSeid; SPFfunctionType fn; SPFargs args);
                                                     $(SPFspecialFunction)
   $(this function will be further specified when its exact nature is known)

END_MODULE
```

WDL-TR7933
Revision 3
December 1980

SECURE MINICOMPUTER OPERATING SYSTEM (KSOS)
UNIX EMULATOR COMPUTER PROGRAM DEVELOPMENT SPECIFICATION (TYPE B5)

Department of Defense Kernelized Secure Operating System

Contract MDA 903-77-C-0333
CDRL 0002AG

Prepared for:

Defense Supply Service - Washington
Room 1D245, The Pentagon
Washington, DC 20310

Ford Aerospace &
Communications Corporation
Western Development
Laboratories Division

3939 Fabian Way
Palo Alto. California 94303

This document is a major revision of an earlier draft dated July 1978. That and other previous editions are obsolete and should not be used.

TABLE OF CONTENTS

Kernelized Secure Operating System (KSOS): UNIX Emulator Computer Program Development Specification (Type B5)

1. SCOPE

1.1 Identification

This specification establishes the performance, design, development and test requirements for the Kernelized Secure Operating System (referred to as "KSOS") UNIX Emulator. The UNIX Emulator is intended to support the user program interface of the Western Electric UNIX operating system Version 6.0.

1.2 Functional Summary

The UNIX Emulator CPCI creates a system call interface which is compatible with that provided by the UNIX operating system [Ritchie 74], [Thompson 75]. The UNIX Emulator is an operating system, running on the "Kernel Machine", which creates UNIX level resources from the more primitive resources provided by the Kernel. The hierarchical UNIX file structure is created through the cooperation between the UNIX Emulator and the trusted Directory Manager. The Emulator also contains the per user aspects of support for the interface to a computer network.

The remainder of these specifications are organized as follows:

- Section 2 contains the references
- Section 3 contains the requirements for the CPCI
- Section 4 contains the quality assurance provisions for the CPCI
- Sections 5 and 6 are not applicable and are null.

These specifications serve two distinct purposes. The first is to define the UNIX Emulator in a generic, machine independent way. The second is to define an implementation for the PDP-11/70. Thus, there are many PDP-11/70 specific remarks. These should be treated as non-binding, informative commentary for non-PDP-11/70 implementations.

1.3 Terminology

The language used throughout this specification attempts to conform to the guidelines of Section 3.2.3 of MIL-STD-490. In particular, the word "shall" means that the specification expresses a provision that is binding. The words "should" and "may" mean that the specification expresses a provision which is non-mandatory. The word "will" is used to express a declaration of purpose on the part of the Government.

KSOS shall support the mandatory DoD security policy of DoD Directive 5200.1-R that is embodied in a Government approved mathematical model. [Verif

78] Proofs of the system's security properties shall be in terms of this model. KSOS shall provide a minimum of eight (8) hierarchical security classifications categories (or simply security categories) and a minimum of thirty-two (32) mutually independent security compartments. The security categories shall be such that

UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP SECRET

Where "<" is defined in accordance with the requirements of DoD 5200.1-R. One security compartment shall be reserved for read protection of system data bases and programs. This compartment shall be called the "system" compartment. KSOS shall provide for Kernel-enforced integrity. Integrity is defined as the formal mathematical dual of security [Biba 75]. At least four (4) hierarchical integrity classifications categories (or simply integrity categories) shall be provided in KSOS. The integrity categories include at least the following three classification categories:

USER < OPERATOR < ADMINISTRATOR

Although there is no immediate requirement for integrity compartments, the development contractor should include provisions to ease their later inclusion. For the remainder of this specification, the term "security level" means the combination of a security category and a set of security compartments (which may be null). The term "integrity level" means the combination of an integrity category, and a (presently always null) set of integrity compartments. The term "level" (or access level) means the security and integrity level, that is, the combination of a security category, a set of security compartments (which may be null), an integrity category, and a (presently always null) set of integrity compartments. The "access matrix" defines for a particular user or group the combination of the maximum security level permitted for that user or group, all security compartments to which that user or group has access, the maximum integrity level for that user or group, and all integrity compartments (presently always null) to which that user or group has access.

In a secure system it is necessary to have processes external to the kernel which perform security critical functions. These have come to be called "trusted processes". In KSOS there is a refinement in this terminology. As Figure 1-1 shows there are subdivisions of the software inside the security perimeter. Some of the processes are "privileged". This means that they may violate one or more of the security rules enforced by the Kernel. These processes must be designed and implemented with the same care and thoroughness as the Kernel itself. A second subdivision has been termed "responsible" processes. These are processes are not privileged to violate any of the Kernel's rules, but which are nonetheless important to the overall security of the system. Examples of responsible processes include the Secure Mail facility and the processes which manipulate the data bases describing the users' security and integrity permissions (the "access matrix"). These processes must also be carefully designed and implemented, but do not require formal specification of verification. Because they do perform security-related functions, they cannot include untrusted components (e.g. the UNIX Emulator). The third subdivision is the "encapsulated utilities" (EU in the figure). These are self contained

```
                      |-------------------------------|
                      |                               |
                      |    User process part          |   User domain
                      |                               |
                      |-----------|-------------      |
                      |           V            |      |
                      | Emulator process part  |      |   Supervisor domain
                      |                        |      |
                      |-----------|----------------|--|
|-----------|         |           V                V  |
|           |         |     Security kernel           |   Kernel domain
| User      |         | ^                             |
|           |         |--|----------------------------|
| Console   |------------                             |
|           |         | Host (PDP-11/70) computer     |
|-----------|         |-------------------------------|
```

Figure 1-1. Terminology Figure

functions to which the user simply supplies parameters. The encapsulated utilities are not privileged to violate Kernel rules, and do not affect the security of users. The chief function in the encapsulated utilities is system generation.

2. APPLICABLE DOCUMENTS

The following documents, of exact issue shown, form a part of this specification to the extent specified herein. In the event of a conflict between the referenced documents and the contents of this specification, this specification shall be considered a superseding requirement. In the text references to these documents are in the form [Name date], e.g. [Biba 75].

2.1 Government Documents

2.1.1 Directives, Manuals and Standards

a. DoD 5200.1-R Information Security Program Regulation

b. DoD 5200.28 Security Requirements for Data Processing (ADP)

c. DoD 5200.28-M ADP Security Manual

d. MIL-STD-483 Configuration Management

e. MIL-STD-490 Specification Practices

f. MIL-STD-1521A Technical Reviews and Audits

2.1.2 Reports

a. [A-Specs 78] "KSOS System Specification (Type A)", WDL-TR7808 Revision 1, Ford Aerospace and Communications Corporation, Palo Alto, CA (July 1978).

b. [Bell and LaPadula 73] Bell, D.E. and LaPadula, L.J., "Secure Computer Systems", ESD-TR-73-278, Volume I-III, MITRE Corporation, Bedford, MA (November 1973 - June 1974).

c. [Biba 75] Biba, K.J., "Integrity Considerations for Secure Computer Systems", MTR-3153, MITRE Corporation, Bedford, MA (June 1975).

d. [Kernel 78] "KSOS Computer Program Development Specification (Type B5): Security Kernel", WDL-TR7932, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

e. [NKSR 78] "KSOS Computer Program Development Specification (Type B5): Non-Kernel Security-Related Software", WDL-TR7934, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

f. [Verif 78] "KSOS Verification Plan", WDL-TR7809, Ford Aerospace and Communications Corporation, Palo Alto, CA (March 1978).

g. [Walter et al. 74] Walter, K.G. et al., "Primitive Models for Computer Security", ESD-TR-74-117, Case Western Reserve University, Cleveland, OH (January 1974).

2.2 Non-Government Documents

a. [BBN 75] "Interface Message Processor, Specifications for the Interconnection of a Host and an IMP", Report 1822, Bolt Beranek and Newman Inc., Cambridge, MA, (December 1975).

b. [Holmgren et al. 77] Holmgren, S.F., Healy, D.C., Jones, P.B. and Kasprzycki, E., "Illinois Inter-Process Communication Facility for UNIX", CAC Technical Memorandum 84, CCTC-WAD Document 7507, Center for Advanced Computation, University of Illinois at Urbana-Champaign, Urbana, IL (April 1977).

c. [Lampson 73] Lampson, B., "A Note on the Confinement Problem", CACM, Volume 16, Number 10, pp 613 - 615 (October 73).

d. [Lipner 75] Lipner, S.B., "A Comment on the Confinement Problem", Proc. Fifth Symposium on Operating Systems Principles, ACM SIGOPS Review, Volume 9, Number 5, pp 192 - 196 (19-21 November 1975).

e. [Nemeth et al. 77] Nemeth, A.G., Sunshine, C., Zucker, S. and Tepper, S., "Progress Towards a DeFacto DoD UNIX Inter-Process Communication Standard", Working Paper, (May 1977). (Available from the authors.)

f. [Parnas 72] Parnas, D.L., "A Technique for Software Module Specification with Examples", CACM, Volume 15, Number 5, pp 330 - 336 (May 1972).

g. [Postel 80a] Postel, J.B., "DOD Standard Internet Protocol", (RFC 760, IEN 128), Information Sciences Institute, University of Southern California, Marina del Rey, CA (January 1980).

h. [Postel 80b] Postel, J.B., "DOD Standard Transmission Protocol", (RFC 761, IEN 129), Information Sciences Institute, University of Southern California, Marina del Rey, CA (January 1980).

i. [Ritchie 74] Ritchie, D.M. and Thompson, K., "The UNIX Timesharing System", CACM, Volume 17, Number 5, pp 365 - 375 (May 1974).

j. [Roubine and Robinson 77] Roubine,O., L.Robinson, Special Reference Manual, 3rd ed., Technical Report CSG-45, SRI International, Menlo Park, CA (January 1977).

k. [Thompson 75] Thompson, K. and Ritchie, D.M., "UNIX Programmer's Manual", Sixth Edition, Western Electric Corporation, Greensboro, NC (May 1975).

l. [Sunshine 77] Sunshine, C., "Interprocess Communication Extensions for the UNIX Operating System: I Design Considerations", R-2064/1-AF, RAND Corporation,

m. [Zucker 77] Zucker, S., "Interprocess Communication Extensions for the UNIX Operating System: II Implementation", R-2064/2-AF, RAND Corporation, Santa Monica, CA (June 1977).

2.3 Other References Not Part of This Specification

a. [UNIX 75] "UNIX Program Listings", Version 6.0, Western Electric Corporation, Greensboro, NC (May 1975).

3. REQUIREMENTS

3.1 Computer Program Definition

This CPCI defines an operating system program to be implemented upon the KSOS Security Kernel which shall emulate the call interface of the Western Electric Corp. UNIX operating system. The resulting operating system shall provide a UNIX environment which enforces DoD military security policy. For additional product specification information, the reader is referred to the KSOS System Specification (Type A) [A-Specs 78].

The KSOS UNIX Emulator has two major components: the operating system call Emulator and the Directory Manager.

3.1.1 Interface Requirements

The UNIX Emulator shall execute in the memory and privilege domain between user programs and the Security Kernel. On the PDP-11/70, the Emulator shall execute in the Supervisor domain of the machine. The Emulator data base shall be protected from tampering by the user process through the virtual memory hardware of the host computer. To the Security Kernel, the Emulator and the user program are considered to be one, untrusted process that simultaneously occupies more than one virtual memory domain.

The user program shall be capable of trapping to either the Emulator or directly to the Security Kernel. In the PDP-11/70 implementation of KSOS, the EMT instruction shall be used to trap to the Kernel. The TRAP instruction shall cause a trap to the Emulator. The Emulator shall be dispatched to the appropriate trap handler, saving the state of the user's registers. The PDP-11/70 KSOS system may allocate general register set 1 to the Emulator and user programs. General register set 0 may be used by the Security Kernel.

The relationship between the user program and the Emulator may be a cooperative one. Because the user may execute Kernel calls directly, it may be possible to circumvent the Emulator and cause undesirable effects. These effects shall be confined to the user process only without disturbing the processes of other system users except as other user processes have arranged and expect to cooperate with the corrupted process (for example through ports or files) and to the extent that the corrupted process deviates from its expected behavior. Although the Emulator data bases shall be protected from direct manipulation by the user program, the Emulator may be viewed as an extension of the user process, acting on the behalf of the user. The user may abuse the services of the Emulator at the peril of the process. Information compromises shall never result from erroneous interaction of the user process part and the Emulator.

3.1.1.1 Interface Block Diagram

Figure 1-2 is the interface block diagram for the KSOS UNIX Emulator. This diagram illustrates the layers of virtual machine or functional capability in the KSOS operating system. The KSOS UNIX Emulator interfaces directly with the Security Kernel and user programs through the interdomain call facility of the PDP-11/70 hardware. The PDP-11 EMT instruction shall be used for direct

```
|--------------------------|----------------|
|                          |                |
|                          |                |
|    User Programs         |                |  User Domain
|                          |                |
|                          |                |
|-------------------|      |    Directory   |
|                   |      |                |
|                   |      |                |
|  UNIX   Emulator  |<---->|     Manager    |  Supervisor Domain
|                   |      |                |
|                   |      |                |
|---------|---------|--|---|--------|-------|
|         V            V            V       |
|                                           |
|           KSOS Security Kernel            |  Kernel Domain
|                                           |
|                                           |
|-------------------------------------------|
```

Figure 1-2. Interface Diagram

Emulator to Kernel communication. The PDP-11 TRAP and IOT instructions shall be used to emulate the UNIX call interface. Existing UNIX software uses the TRAP and IOT instructions for operating system calls. Hence, compatibility with current UNIX shall be maintained. The UNIX call Emulator spawns a directory manager as needed. Directory creation, deletion and update requests shall be transfered through a parameter segment when the Directory Manager is spawned by the Emulator. Responses from the Directory Manager (i.e., error reports) shall be sent by IPC message to the Emulator of the process which initiated the directory operation.

3.1.1.2 Detailed Interface Definition

3.1.1.2.1 Files

UNIX files shall remain intact in KSOS. The user shall be restricted from observing certain file status information, i.e., link counts. These restrictions shall be required to remove clandestine information channels between users of different security and integrity levels.

The Security Kernel shall support a mutual exclusion mechanism on Kernel level files. (This mechanism may be used to regulate updates on directories.) Future implementations of the KSOS Emulator should be able to take advantage of the open for exclusive use feature of the Kernel. When a file is opened for exclusive use, other processes attempting to open the file for reading or writing shall be forced to block until the file has been closed or until a system default timeout has expired. Support of exclusive use files at the KSOS user level is beyond the scope of the current Emulator contract, but would be a rewarding future addition to the Emulator.

3.1.1.2.2 Directories

UNIX directory semantics shall be retained in their entirety in KSOS. The operation of the UNIX calls link, unlink and creat shall be compatible with standard UNIX relative to directory semantics. (However, the access checking on link and unlink shall be tightened in KSOS.)

The integrity of the KSOS file systems should be improved over the present Version 6.0 UNIX. Directories may be implemented using the Kernel file subtype mechanism, preventing casual manipulation of file directories. The Directory Manager shall provide a substantial demonstration of KSOS file subtypes. The KSOS user shall be unaware of the existence of subtypes; the Emulator shall make the directory mechanism transparent to the user process.

3.1.1.2.3 Processes

Standard UNIX process semantics shall also be retained. Due to the mediation of the Kernel, the invocation of trusted software and core dumping shall be secure. Trusted software shall be invoked through the Kernel K_invoke and K_spawn calls. Because core dumps shall be performed outside the security perimeter of the Kernel by the Emu'ator, unauthorized disclosure of information shall be prevented. The Emulator shall not write a core dump file without the appropriate access capability. Communication with trusted children shall be disallowed by the child, preventing compromises through ptrace and other interprocess communication (IPC) channels.

KSOS users may be capable of sharing data through common process segments. Process segments shall be manipulated through the direct user to Kernel interface. Future enhancements to the Emulator should provide Emulator calls for manipulating segments, preventing Emulator "blind spots" and providing a more congenial interface to the user for segment manipulation. Although the user may succeed in confusing the Emulator by changing process segments directly, no compromise of information shall occur.

3.1.1.2.4 Pipes

Version 6.0 pipes shall be implemented by the KSOS Emulator. Pipes may be implemented using Kernel-level shared segments.

3.1.1.2.5 Ports

Ports will be a user level mechanism for inter-process communication. KSOS ports will be based upon RAND ports, augmented by two new operating system calls (await and capac) which provide low level channel synchronization [Nemeth et al. 77]. Ports may be implemented using shared segment communications, providing greater channel bandwidth than current implementations. Further information on RAND ports may be found in [Sunshine 77] and [Zucker 77].

abort	3.2.1.	IOT
await	3.2.2.	59
break	3.2.3.	17
capac	3.2.4.	63
chdir	3.2.5.	12
chmod	3.2.6.	15
chown	3.2.7.	(Subsumed into NKSR software)
close	3.2.8.	6
creat	3.2.9.	8
csw	3.2.10.	(unsupported)
dup	3.2.11.	38
eofp	3.2.12.	60
exec	3.2.13.	11
exit	3.2.14.	1
fork	3.2.15.	2
fstat	3.2.16.	28
getal	3.2.17.	58
getgid	3.2.18.	47
getpid	3.2.19.	20
getuid	3.2.20.	24
gprocs	3.2.21.	61
gtty	3.2.22.	32
indir	3.2.23.	0
kill	3.2.24.	37
link	3.2.25.	9
mknod	3.2.26.	14
mount	3.2.27.	(Subsumed into NKSR software)
nice	3.2.28.	34
open	3.2.29.	5
pipe	3.2.30.	42
port	3.2.31.	57
profil	3.2.32.	44
ptrace	3.2.33.	26
read	3.2.34.	3
seek	3.2.35.	19
setgid	3.2.36.	46
setuid	3.2.37.	23
sfork	3.2.38.	62
signal	3.2.39.	48
sleep	3.2.40.	35
stat	3.2.41.	18
stime	3.2.42.	(Subsumed into NKSR Software)
stty	3.2.43.	31
sync	3.2.44.	36
time	3.2.45	13
times	3.2.46.	43
umount	3.2.47.	(Subsumed into NKSR Software)
unlink	3.2.48.	10
wait	3.2.49.	7
write	3.2.50.	4

Figure 3-3. List of KSOS UNIX Emulator Calls

3.2 Detailed Functional Requirements

This paragraph describes the detailed functional requirements of the KSOS UNIX Emulator. Figure 3-2 summarizes the Emulator calls by name (listed alphabetically), Specification paragraph number, and system entry number. The system entry number corresponds to the low order byte of the PDP-11/70 TRAP instruction used to enter the Emulator.

For each call, a list of exception conditions is provided. The order of this list is not necessarily the order in which the conditions would be evaluated. The development contractor may alter the order or add to the exception conditions as needed, subject to Government approval. Various exceptions detected during a system call may not return distinct error numbers, but rather, may maintain compatibility with existing UNIX error numbers.

3.2.1 abort

3.2.1.1 Inputs

The Emulator call abort shall take the following parameters.

None

3.2.1.2 Processing

The Emulator abort call shall cause a core image of the user-mode part of the process and the system's per-user data for that user to be dumped to a disk file. The core image file shall not be produced unless all the conditions necessary to create a file in the working directory of the process have been satisfied, including mandatory security and discretionary access permissions. (These same conditions shall apply to dumps created as a result of some process induced fault.)

3.2.1.3 Outputs

The Emulator call abort shall return the following values after execution.

None (Does not return)

The core file shall be created, if possible, and the parent process shall be notified about the death of this process.

3.2.2 await

3.2.2.1 Inputs

The Emulator call await shall take the following parameters.

to: time out
wc: wake up conditions

3.2.2.2 Processing

(Not present in Version 6.0 UNIX.) The Emulator call await shall permit a process to suspend its execution until either a time out event has occurred or one of the specified wake up conditions has been met. The time out value may be specified in 1/60 seconds, or some other appropriate time unit. The minimum set of wake up conditions to be implemented shall be:

a. wake up when any pipe is written

b. wake up when any pipe is read

c. wake up when any pipe becomes empty

If the desired wake up event has occurred at the time of the await call, the process shall not block, but return to the process immediately. The wake up value zero shall cause the process to wait only on the wake up conditions.

3.2.2.3 Outputs

The Emulator call await shall return the following values after execution.

cw: wake up event which caused process to awaken
ec: error conditions

The Emulator call await shall detect and report the following exception (error) conditions.

EX: invalid time out value
EX: invalid wake up condition specified

3.2.3 break

3.2.3.1 Inputs

The Emulator call break shall take the following parameters.

na: new break address

3.2.3.2 Processing

The Emulator call break shall increase the length of the process data segment to the new break address. Break shall also permit process data segments to shrink. Access attempts to locations between the break address and the stack pointer shall cause a segmentation violation signal if access is attempted to a currently unmapped portion of the virtual address space (PDP-11/70 peculiarity). The PDP-11/70 implementation of KSOS shall express the break address as a multiple of 512 bytes.

3.2.3.3 Outputs

The Emulator call break shall return the following values after execution.

ec: error conditions

The Emulator call break shall detect and report the following exception (error) conditions.

EX: physical memory resources exhausted
EX: memory management hardware resources exhausted

3.2.4 capac

3.2.4.1 Inputs

The Emulator call capac shall take the following parameters.

fd: file descriptor of pipe (or port)
cv: pointer to two integer vector for byte counts

3.2.4.2 Processing

(Not present in Version 6.0 UNIX.) (This call subsumes the RAND empty call [Sunshine 77] and [Zucker 77].) The capac Emulator call shall return the number of bytes in a pipe (or port) available for reading and writing. This call shall not cause the process to block.

3.2.4.3 Outputs

The Emulator call capac shall return the following values after execution.

br: number of bytes available for reading
bw: number of bytes available for writing

The Emulator call capac shall detect and report the following exception (error) conditions.

EX: invalid file descriptor
EX: file descriptor does not refer to pipe (or port)
EX: improper vector pointer address

3.2.5 chdir

3.2.5.1 Inputs

The Emulator call chdir shall take the following parameters.

pn: pathname of directory file

3.2.5.2 Processing

The Emulator call chdir shall change the current process working directory to the specified directory.

3.2.5.3 Outputs

The Emulator call chdir shall return the following values after execution.

ec: error conditions

The Emulator call chdir shall detect and report the following exception (error) conditions.

EX: directory required to interpret pathname is unmounted
EX: file on name interpretation path is not a directory
EX: directory on name interpretation path does not exist
EX: directory on name interpretation path has search permission denied
EX: directory required for name interpretation has higher security level (returned as if directory did not exist)
EX: directory on name interpretation path is at lower integrity level (returned as if directory did not exist)

3.2.6 chmod

3.2.6.1 Inputs

The Emulator call chmod shall take the following parameters.

pn: pointer to file pathname
da: new discretionary access mode

3.2.6.2 Processing

The chmod call shall change the discretionary access mode of a file. Only the file owner and sufficiently privileged processes may change the discretionary access mode of a file. Because the chmod call is partially processed by the untrusted Emulator, it is conceivable, though unlikely, that there would be a "trojan horse" embedded in the Emulator that would not actually perform the requested chmod. For example, the trojan horse could always set the protection to 777. If the user wishes to be absolutely sure that the chmod is done only by trusted software, the File Access Modifier [NKSR 78] may be employed at command level. These remarks apply only to the discretionary access information. The hypothetical trojan horse could not affect the security level of the file.

3.2.6.3 Outputs

The Emulator call chmod shall return the following values after execution.

ec: error conditions

The Emulator call chmod shall detect and report the following exception (error) conditions.

EX: directory needed for name interpretation is on unmounted file system
EX: file on name interpretation path is not a directory
EX: directory on name interpretation path does not exist
EX: directory on name interpretation path has search permission denied
EX: directory on interpretation path is at higher security level (returned as if directory did not exist)
EX: directory on interpretation path is at lower integrity level (returned as if directory did not exist)
EX: file name could not be found in directory
EX: file is at lower security level than process (returned as if file did not exist)
EX: file is at higher integrity level than calling process (returned as if file did not exist)
EX: file does not exist

3.2.7 chown

The standard UNIX call chown has been subsumed into the File Access Modification process of the Non-Kernel Security Related Software CPCI.

3.2.8 close

3.2.8.1 Inputs

The Emulator call close shall take the following parameters.

fd: file descriptor

3.2.8.2 Processing

The close call shall terminate input/output to the specified file, device, or pipe. A file descriptor shall specify the file to be closed. In the PDP-11/70 implementation, the file descriptor shall be passed in register R0. The last close of a file whose link count has decremented to zero shall cause the file to be deleted. Termination of a process shall implicitly close all of its open files.

3.2.8.3 Outputs

The Emulator call close shall return the following values after execution.

ec: error conditions

The Emulator call close shall detect and report the following exception (error) conditions.

EX: invalid file descriptor

3.2.9 creat

3.2.9.1 Inputs

The Emulator call creat shall take the following parameters.

pn: pointer to user pathname string in user domain
da: discretionary access mode of new file

3.2.9.2 Processing

The Emulator call creat shall create a new file. The file shall be subsequently accessible by

a. the completely specified (rooted) pathname, or

b. a concatenation of the process working directory string and the specified pathname string.

The file shall be created with the specified discretionary access. (See the remark in chmod about discretionary access.) The parent directory of the file shall be updated to include the new file. The newly created file shall have a security and integrity level equal to that of the calling process.

If the file exists, but is at a security level where it cannot be written, the error return shall be as if discretionary access to it was denied. This preserves the UNIX construct of lock files.

An open file descriptor shall be returned by creat. This descriptor shall be used to reference the file in subsequent operations. The file shall be opened for writing. In the PDP-11/70 implementation, the file descriptor shall be returned in register R0.

3.2.9.3 Outputs

The Emulator call creat shall return the following values after execution.

fd: open file descriptor
ec: error conditions

The Emulator call creat shall detect and report the following exception (error) conditions.

EX: security or integrity level not permitted in file system upon which the file would be created
EX: file system required for pathname interpretation is not mounted
EX: file name required for pathname interpretation is not a directory
EX: directory in name interpretation path is unsearchable or does not exist
EX: file to be created exists and is a directory
EX: directory to be written is not writably mounted
EX: file does not exist and write permission on directory is denied
EX: file does exist and write permission on it is denied
EX: too many open files

3.2.10 csw

The standard UNIX operating system call csw will not be supported by the KSOS Kernel.

3.2.11 dup

3.2.11.1 Inputs

The Emulator call dup shall take the following parameters.

fd: file descriptor

3.2.11.2 Processing

The dup call shall allocate and return another file descriptor which is synonymous with the original descriptor. The assignment algorithm shall return the lowest value available.

3.2.11.3 Outputs

The Emulator call dup shall return the following values after execution.

fd: new file descriptor
ec: error conditions

The Emulator call dup shall detect and report the following exception (error) conditions.

EX: invalid file descriptor
EX: no free file descriptors

3.2.12 eofp

3.2.12.1 Inputs

The Emulator call eofp shall take the following parameters.

fd: file descriptor of pipe

3.2.12.2 Processing

(Not present in Version 6.0 UNIX.) The Emulator call eofp shall write an end of file on a pipe. Additional writes may be performed on the pipe, but they are suspended until the eof has been read. If the eofp call has been issued, after reading all the valid data, the next read on the pipe shall return 0 characters.

3.2.12.3 Outputs

The Emulator call eofp shall return the following values after execution.

ec: error conditions

The Emulator call eofp shall detect and report the following exception (error) conditions.

EX: invalid file descriptor or file descriptor does not refer to a pipe

3.2.13 exec

3.2.13.1 Inputs

The Emulator call exec shall take the following parameters.

na: pointer to file name string of process prototype image
ap: pointer to zero terminated list of argument strings

3.2.13.2 Processing

The Emulator call exec shall overlay the current user domain process image with a new image as defined by the process prototype file. The calling process image shall be lost. Open files shall remain open after the execution of the exec call. Signal function values shall be set to the defaults, except for ignored signals, which remain ignored. Profiling shall be disabled by the execution of exec. If the process image to be incarnated is privileged (i.e., privilege bits in the type independent information for the file are enabled) the effective process user ID shall be updated accordingly. Reference [Thompson 75] section A.OUT(V) for the definition of the process prototype file format.

3.2.13.3 Outputs

The Emulator call exec shall return the following values after execution.

ec: error conditions

The Emulator call exec shall detect and report the following exception (error) conditions.

EX: security level of process is too low to execute (read) file (returned as if file did not exist)
EX: integrity level of process is too high to execute (read) file (returned as if file did not exist)
EX: process prototype file does not exist
EX: process user ID does not have execute permission to prototype file
EX: too many arguments (argument list too large) The limit on the size of the argument segment shall be no smaller than the present limit (approximately 512 characters).
EX: file prototype format error
EX: not enough memory resources to execute process image

3.2.14 exit

3.2.14.1 Inputs

The Emulator call exit shall take the following parameters.

es: exit status value

3.2.14.2 Processing

The Emulator call exit shall cause the process (both user and Emulator parts) to terminate execution. All open files shall be closed. The parent process shall be notified of the process termination. The exit status value shall be returned to the parent as part of the notification. The exit call shall never return.

3.2.14.3 Outputs

The Emulator call exit shall return the following values after execution.

None

The Emulator call exit shall detect and report the following exception (error) conditions.

None

3.2.15 fork

3.2.15.1 Inputs

The Emulator call fork shall take the following parameters.

None

3.2.15.2 Processing

The Emulator call fork shall create a new process. The new process image shall be identical to the parent (original) process image. Open files shall be passed to the child (newly created) process. The child process shall receive the signal function values of the parent process. If profiling is enabled in the parent process, profiling shall be inherited by the child. The return value of the parent process shall be the process ID of the child process. The child process shall receive the return value zero. In the PDP-11/70 implementation of KSOS, the program counter of the parent process shall be incremented by two, causing the parent to return to a different location after the execution of fork. The clandestine information channel offered by the resource exhaustion condition shall be bandwidth limited by the KSOS Security Kernel.

3.2.15.3 Outputs

The Emulator call fork shall return the following values after execution.

pi: process id of child (returned to parent process), zero (returned to child process)

The Emulator call fork shall detect and report the following exception (error) conditions.

EX: process space exhausted

3.2.16 fstat

3.2.16.1 Inputs

The Emulator call fstat shall take the following parameters.

fd: file descriptor (passed in register R0)
ba: buffer address for file status information

3.2.16.2 Processing

The fstat call shall return the same information (except for shared values like link counts that could be used as channels) and in the same format as the stat call below. The file descriptor shall refer to a presently open (read or read/write mode) file. In KSOS, a file must be readable with respect to mandatory security before the control information can be accessed to prevent clandestine information channels.

3.2.16.3 Outputs

The Emulator call fstat shall return the following values after execution.

fs: file status information
ec: error conditions

The Emulator call fstat shall detect and report the following exception (error) conditions.

EX: invalid file descriptor
EX: file has not been opened for reading

3.2.17 getal

3.2.17.1 Inputs

The Emulator call getal shall take the following parameters.

bp: pointer to buffer for access level information

3.2.17.2 Processing

(Not present in Version 6.0 UNIX.) The getal call shall return the type independent information for the process: security classification, security category, integrity level, integrity category, and effective/real group and user identifiers. The user identification information is returned in the full bit-precision supported by the Security Kernel. The user identification returned by the getuid and getgid calls represents the shorter "mapped" identifiers as transformed by the Emulator.

3.2.17.3 Outputs

The Emulator call getal shall return the following values after execution.

al: process access level information

The Emulator call getal shall detect and report the following exception (error) conditions.

EX: invalid buffer pointer address

3.2.18 getgid

3.2.18.1 Inputs

The Emulator call getgid shall take the following parameters.

None

3.2.18.2 Processing

The Emulator call getgid shall return the effective and real group ID of the process. In the PDP-11/70 implementation of KSOS, the high order byte of R0 shall contain the effective ID. The low order byte shall contain the real ID. The real ID shall identify the user who initiated execution of the process.

The KSOS Security Kernel supports user and group ID's which are longer than the 8-bits implemented in standard UNIX. The UNIX Emulator shall map longer Kernel names to shorter UNIX compatible identifiers. (This note shall also apply to the Emulator call getuid, below.)

3.2.18.3 Outputs

The Emulator call getgid shall return the following values after execution.

er: effective and real group ID of process

The Emulator call getgid shall detect and report the following exception (error) conditions.

None

3.2.19 getpid

3.2.19.1 Inputs

The Emulator call getpid shall take the following parameters.

None

3.2.19.2 Processing

The Emulator call getpid shall return the process ID of the calling process. This name may be a mapped version of the longer Kernel names for processes (SEID's).

3.2.19.3 Outputs

The Emulator call getpid shall return the following values after execution.

pi: process ID of calling process

The Emulator call getpid shall detect and report the following exception (error) conditions.

None

3.2.20 getuid

3.2.20.1 Inputs

The Emulator call getuid shall take the following parameters.

None

3.2.20.2 getuid

The Emulator call getuid shall return the effective and real user ID of the calling process. In the PDP-11/70 implementation of KSOS, the low order byte of R0 shall contain the real user ID. The high order byte shall return the effective user ID. See the notes in getgid above for a discussion of the uses of real and effective ID.

3.2.20.3 Outputs

The Emulator call getuid shall return the following values after execution.

er: effective and real user ID of calling process

The Emulator call getuid shall detect and report the following exception (error) conditions.

None

3.2.21 gprocs

3.2.21.1 Inputs

The Emulator call gprocs shall take the following parameters.

ba: buffer address for process status information (in register R0)

3.2.21.2 Processing

The gprocs call shall copy process control information to a buffer in the calling user process domain. The process control data shall be at a lower or equal security level than the caller. The number of observed processes shall be returned. Only information about processes in the process family of the caller shall be returned. In the PDP-11/70 implementation the returned information shall adhere to the format of the existing UNIX process table structure to the extent possible. Security relevant information shall not be returned, and KSOS relevant information may be returned instead.

3.2.21.3 Outputs

The Emulator call gprocs shall return the following values after execution.

np: number of process table entries returned to the user process
ec: error conditions

The Emulator call gprocs shall detect and report the following exception (error) conditions.

EX: bad buffer address

3.2.22 gtty

3.2.22.1 Inputs

The Emulator call gtty shall take the following parameters.

fd: open file descriptor
sb: pointer to data vector in user domain for status information

3.2.22.2 Processing

The Emulator call gtty shall return the current status of a device currently open and accessible for reading by the calling process.

3.2.22.3 Outputs

The Emulator call gtty shall return the following values after execution.

ec: error conditions
si: device status information

The Emulator call gtty shall detect and report the following exception (error) conditions.

EX: invalid file descriptor
EX: device is not a character mode device

3.2.23 indir

3.2.23.1 Inputs

The Emulator call indir shall take the following parameters.

sc: pointer to location containing a system call

3.2.23.2 Processing

The indir Emulator call shall execute the system call specified by the pointer argument. If indir is executed indirectly, no effect shall occur. Attempts to execute non-system calls shall result in a process fault.

3.2.23.3 Outputs

The Emulator call indir shall return the following values after execution.

ec: error conditions

The Emulator call shall detect and report the following exception (error) conditions.

EX: attempt to execute non-system call
EX: invalid address

3.2.24 kill

3.2.24.1 Inputs

The Emulator call kill shall take the following parameters.

pi: process ID of destination process
sn: number of signal to be sent

3.2.24.2 Processing

The Emulator call kill shall send the specified signal to the destination process. It shall not be possible for a process to kill itself. In the PDP-11/70 implementation of KSOS, the process number shall be passed in register R0. An error shall be detected and returned if the destination process has denied IPC writes via setting the discretionary access information for the process.

3.2.24.3 Outputs

The Emulator call kill shall return the following values after execution.

ec: error conditions

The Emulator call kill shall detect and report the following exception (error) conditions.

EX: destination process is the calling process
EX: destination process has denied IPC writes to it (returned as if the process did not exist)
EX: destination process does not exist
EX: signal number is less than zero
EX: signal number is greater than the Emulator maximum

3.2.25 link

3.2.25.1 Inputs

The Emulator call link shall take the following parameters.

dn: pathname of file to which the link shall be made
ln: pathname of the link

3.2.25.2 Processing

The link Emulator call shall create a link to a file with the specified pathname. The pathname interpretation and verification algorithm shall be applied to both specified pathnames. The full semantics of UNIX links shall be retained in KSOS.

3.2.25.3 Outputs

The Emulator call link shall return the following values after execution.

ec: error conditions

The Emulator call link shall detect and report the following exception (error) conditions.

EX: file system required for name interpretation is not mounted
EX: file required for name interpretation is not a directory
EX: directory on path is at a higher security level (returned as if the directory did not exist)
EX: directory on path is at a lower integrity level (returned as if the directory did not exist)
EX: directory on path has search permission denied
EX: link name already exists
EX: security level of directory to be written is higher than caller (returned as if there were no write permission on the directory)
EX: integrity level of directory to be written is lower than caller (returned as if there were no write permission on the directory)
EX: write permission to directory has been denied
EX: attempt to link across file systems

3.2.26 mknod

3.2.26.1 Inputs

The Emulator call mknod shall take the following parameters.

pn: pointer to new file pathname
da: discretionary access mode of new file
ad: "address" field (retained for UNIX compatibility)

3.2.26.2 Processing

The Emulator call mknod shall create a new file with the specified pathname. The discretionary access mode of the new file shall be set to the specified mode. (See the comments on chmod.) A directory shall (in all cases) be created. The creation of special (device) files shall be accomplished through system generation and NKSR Software functions. Unlike standard UNIX, regular user processes shall be able to create directories. Regular user processes shall not be able to write directly to directories as regulated by the Kernel subtype mechanism.

3.2.26.3 Outputs

The Emulator call mknod shall return the following values after execution.

ec: error conditions

The Emulator call mknod shall detect and report the following exception (error) conditions.

EX: insufficient integrity level to create a special file
EX: file in the searchable path is not a directory
EX: file system required for pathname interpretation is not mounted
EX: file exists with the same pathname

3.2.27 mount

The mount UNIX call has been replaced by the mount mechanism of the Non-Kernel Security-Related Software which has the same functionality.

3.2.28 nice

3.2.28.1 Inputs

The Emulator call nice shall take the following parameters.

np: new process priority (passed in R0)

3.2.28.2 Processing

The Emulator call nice shall advise the scheduler of a new runtime priority. This new priority shall only be an advisory. That is, it shall be the responsibility of the scheduler to guarantee equitable service. The legal values which the new priority may assume shall be greater than zero and less than some maximum, currently twenty (20.)

3.2.28.3 Outputs

The Emulator call nice shall return the following values after execution.

ec: error conditions

The Emulator call nice shall detect and report the following exception (error) conditions.

EX: new priority is less than zero
EX: new priority greater than system maximum

3.2.29 open

3.2.29.1 Inputs

The Emulator call open shall take the following parameters.

pn: pointer to pathname string of file to be opened
md: requested access mode { read, write, read and write}

3.2.29.2 Processing

The Emulator call open shall open the designated file for reading and/or writing. A file descriptor shall be returned for use in subsequent references to the file.

3.2.29.3 Outputs

The Emulator call open shall return the following values after execution.

fd: file descriptor (returned in register R0)
ec: error conditions

The Emulator call open shall detect and report the following exception (error) conditions.

EX: file required for name interpretation is not a directory
EX: file system required for name interpretation is not mounted
EX: directory on path is at higher security level (returned as if the directory did not exist)

EX: directory on path is at lower integrity level (returned as if the directory did not exist)
EX: search permission on a pathname directory is denied
EX: file is at a higher security level and reading is requested (returned as if the file has read access denied)
EX: file is at a lower security level and writing is requested (returned as if the file had write access denied)
EX: file is at a lower integrity level and reading is requested (returned as if the file had read access denied)
EX: file is at a higher integrity level and writing is requested (returned as if the file has write access denied)
EX: discretionary permission is denied for writing, write requested
EX: discretionary permission is denied for reading, reading requested
EX: too many files are open

3.2.30 pipe

3.2.30.1 Inputs

The Emulator call pipe shall take the following parameters.

None

3.2.30.2 Processing

The Emulator call pipe shall create an input/output pipe. Two open file descriptors shall be returned, one for input and a second for output. In the PDP-11/70 implementation, the read file descriptor shall be returned in register R0. The write file descriptor shall be returned in register R1. Two available open file slots shall be required to successfully perform the pipe call.

3.2.30.3 Outputs

The Emulator call pipe shall return the following values after execution.

rf: read file descriptor
wf: write file descriptor

The Emulator call pipe shall detect and report the following exception (error) conditions.

EX: too many files open

3.2.31 port

3.2.31.1 Inputs

The Emulator call port shall take the following parameters.

pn: port pathname string
mo: access mode for port
pf: pointer to a two integer block for returned file descriptors

3.2.31.2 Processing

(Not present in Version 6.0 UNIX.) Ports are a generalization of pipes. The port Emulator call shall create and open a new port. The specified pathname shall be either a complete pathname, a pathname relative to the process working directory, or the null string. Ports with the null string as their name shall be inaccessible by other processes. The port shall be at the same security and integrity level of the calling process. The mode parameter shall determine the format of the message headers attached at port data during write operations. The mode parameter shall also specify the write access to the port accorded to the creator, the creator's group and all other users. ([Zucker 77] and [Sunshine 77] discuss the supporting concepts and mechanization of ports in RAND UNIX.)

3.2.31.3 Outputs

The Emulator call port shall return the following values after execution.

fr: file descriptor for reading (also returned in register R0)
fw: file descriptor for writing

The Emulator call port shall detect and report the following exception (error) conditions.

EX: improper mode
EX: directory needed to interpret pathname is unmounted
EX: file on name interpretation path is not a directory
EX: search permission denied for directory on interpretation path
EX: attempted violation of mandatory security during name interpretation (returned as if directory had search permission denied)
EX: invalid pointer address for file descriptors
EX: port already exists

3.2.32 profil

3.2.32.1 Inputs

The Emulator call profil shall take the following parameters.

bp: pointer to histogram buffer in user domain
bs: buffer size (in bytes)
os: offset to be subtracted from sampled program counter
sc: scale to expand/contract observation

3.2.32.2 Processing

The Emulator call profil shall cause an execution profile to be compiled into a data vector in the user process domain. The user program counter will be sampled each 1/10 of a second thereafter while the process is physically executing. Offset and scale shall be used to adjust the program counter value to an index within the bounds of the histogram vector. If the resulting value is within the histogram vector, the observation is tallied. See [Thompson 75] for data types and formats.

In the PDP-11/70 implementation of KSOS, profiling shall be disabled by specifying a scale of zero or one. Profiling shall be rendered ineffective if the buffer size is zero. The execution of the call exec will disable profiling. Profiling shall remain enabled in both parent and child after a fork call.

3.2.32.3 Outputs

The Emulator call profil shall return the following values after execution.

ht: histogram tallies in the data vector
ec: error conditions

The Emulator call profil shall detect and report the following exception (error) conditions.

EX: improper buffer address in user process (not checked in 6.0 UNIX.)
EX: buffer size too large (not checked in 6.0 UNIX.)

3.2.33 ptrace

3.2.33.1 Inputs

The Emulator call ptrace shall take the following parameters.

rq: requested trace action
pi: process ID of traced (destination) process
ad: address parameter to action
da: data parameter to requested action

3.2.33.2 Processing

The Emulator call ptrace shall permit a parent process to control and monitor the execution of an immediate child process. Tracing of set-uid software shall not be permitted. The behavior of the ptrace call shall depend upon the interpretation of the requested action parameter. In the PDP-11/70 implementation of KSOS, the following action values and actions shall be supported.

0. executed by child. indicates that child expects to be traced.
1,2. get instruction space or data space word, respectively, from the child process.

3. returned the addressed word from the per-process data table.
4,5. set word in the child instruction or data space, respectively.
6. write the per-process data word as addressed.
7. force a signal to the child process.
8. terminate the traced process.

With the exception of request zero (0), the traced process shall be stopped prior to execution of the tracing action. The wait call shall be used to determine when a process has stopped. Processes that have stopped shall return to the waiting parent the termination status value of 0177 (octal.)

Because ptrace may be implemented using the low bandwidth IPC mechanism, the usual security, integrity, and discretionary controls enforced by the Kernel may limit the extent of process tracing.

3.2.33.3 Outputs

The Emulator call ptrace shall return the following values after execution.

dv: data values
ec: error conditions

The Emulator call ptrace shall detect and report the following exception (error) conditions.

EX: request value is out of bounds (less than 0 or more than 8)
EX: requested action zero and parent does not exist
EX: destination process does not exist
EX: destination process has denied IPC writes to it (returned as if process did not exist)
EX: destination process is not an immediate descendant of caller
EX: requested data read or write and address was odd
EX: requested data read or write and addressing error resulted

3.2.34 read

3.2.34.1 Inputs

The Emulator call read shall take the following parameters.

fd: file descriptor (passed in register R0)
ba: pointer to buffer for incoming data
nb: number of bytes to be read (if possible)

3.2.34.2 Processing

The read call shall transfer data from the specified file, device, or pipe to a buffer in the process user domain. The number of bytes specified in the call shall be transfered if possible. The actual number of bytes read shall be returned. In the PDP-11/70 implementation, the actual byte count shall be returned in register R0. The value zero shall be returned when end of file has been reached.

3.2.34.3 Outputs

The Emulator call read shall return the following values after execution.

db: data bytes to the user domain buffer
ac: actual count of data bytes transfered

The Emulator call read shall detect and report the following exception (error) conditions.

EX: invalid file descriptor
EX: file has not been opened for reading
EX: attempt to read from pipe without a writer at the other end
EX: physical device error
EX: illegal buffer address

3.2.35 seek

3.2.35.1 Inputs

The Emulator call seek shall take the following parameters.

fd: file descriptor (passed in register R0)
os: logical offset within file
pr: requested pointer adjustment action

3.2.35.2 Processing

The Emulator call seek shall adjust the logical file pointer for the designated open file. The pointer shall be adjusted according to the requested action as follows.

0: the pointer shall be set exactly to the parameter offset.
1: the pointer shall be set to the current seek pointer value plus the offset.
2: the pointer is set to the length of the file plus the offset.
3,4,5: the pointer shall be set as in 0,1,2 above but offset shall be multiplied by 512 before updating the pointer.

Absolute offset values shall be unsigned (case 0 and 3 above.) The offset shall otherwise be signed.

3.2.35.3 Outputs

The Emulator call seek shall return the following values after execution.

ec: error conditions

The Emulator call seek shall detect and report the following exception (error) conditions.

EX: invalid file descriptor
EX: file descriptor refers to a pipe
EX: requested action is out of bounds (pr < 0 or pr > 5)

3.2.36 setgid

3.2.36.1 Inputs

The Emulator call setgid shall take the following parameters.

gi: new group id

3.2.36.2 Processing

The Emulator call setgid shall set the effective group id of the calling process. Processes which are appropriately privileged shall also set the real group id (i.e., the owner) of the process. Unprivileged processes shall be permitted to set the effective group id to the value of the real group id only.

3.2.36.3 Outputs

The Emulator call setgid shall return the following values after execution.

ec: error conditions

The Emulator call setgid shall detect and report the following exception (error) conditions.

EX: attempt to set effective group id to value other than real group id

3.2.37 setuid

3.2.37.1 Inputs

The Emulator call setuid shall take the following parameters.

nu: new effective user id

3.2.37.2 Processing

The Emulator call setuid shall set the effective user id of the calling process. Processes which are appropriately privileged shall also set the real user id (i.e., the owner) of the process. Unprivileged processes shall be permitted to set the effective user id to the value of the real user id only.

3.2.37.3 Outputs

The Emulator call setuid shall return the following values after execution.

ec: error conditions

The Emulator call setuid shall detect and report the following exception (error) conditions.

EX: attempt to set effective user id to value other than real user id

3.2.38 sfork

3.2.38.1 Inputs

The Emulator call sfork shall take the following parameters.

None

3.2.38.2 Processing

The sfork call shall spawn a new process. The process creation function of this call shall be identical to the Emulator call fork. (See "fork" above.) The children generated by this call shall be immune from terminal generated interrupt (DEL character) or quit (FS character) signals. The children shall still receive direct signals from the kill Emulator call. (See "kill" above.)

3.2.38.3 Outputs

The Emulator call sfork shall return the following values after execution.

Identical to fork

The Emulator call sfork shall detect and report the following exception (error) conditions.

Identical to fork

3.2.39 signal

3.2.39.1 Inputs

The Emulator call signal shall take the following parameters.

sn: signal number
sf: signal function

3.2.39.2 <u>Processing</u>

The Emulator call signal shall permit the user to specify the recovery action to be taken following the reception of a signal from another process. The PDP-11/70 implementation of KSOS shall provide for a minimum of thirteen (13) signals. The signal conditions shall be:

1. hangup
2. interrupt
3. quit (core image)
4. illegal instruction (not reset when caught - core image)
5. trace trap (not reset when caught - core image)
6. IOT instruction (core image)
7. EMT instruction (Kernel call - effect differs from standard UNIX)
8. floating point exception (core image)
9. kill (cannot be caught or ignored)
10. bus error (core image)
11. segmentation violation (core image)
12. bad argument to system call (core image)
13. write on a pipe with no one to read it

Note the difference from standard UNIX in signal number seven (7.) The user part of a process shall be able to directly call the Security Kernel. Because the EMT instruction shall be used to invoke the Kernel, the semantics of signal number seven must change. This signal slot shall be retained for historical reasons. Core dump images shall be generated for the signals so marked above.

The PDP-11/70 implementation of KSOS shall comply with the standard UNIX signal semantics relevant to the significance of signal function values and signal routine return procedures. If the signal function value is zero, the process shall terminate when the specified signal is received. This shall be the default action. If the signal function value is odd, the signal shall be ignored except where noted in the list above. An even valued signal function shall specify the address in the user domain of the signal handling routine. This routine shall be entered following the reception of the specified signal. A PDP-11/70 RTI or RTT instruction shall return from the signal interrupt. Signals which have been caught shall cause the selected action to be reset to zero. The number of signals to be set shall be a parameter to the Emulator generation procedure. The number shall not be less than thirteen.

3.2.39.3 <u>Outputs</u>

The Emulator call signal shall return the following values after execution.

ec: error conditions
ov: old signal value

The Emulator call signal shall detect and report the following exception (error) conditions.

EX: specified signal number is less than zero
EX: specified signal number is greater than the maximum number of signals permitted by the Emulator

EX: specified signal is the kill signal (number 9)

3.2.40 sleep

3.2.40.1 Inputs

The Emulator call sleep shall take the following parameters.

st: sleep time in seconds

3.2.40.2 Processing

The Emulator call sleep shall suspend execution for the time period specified by the parameter.

3.2.40.3 Outputs

The Emulator call sleep shall return the following values after execution.

None

The Emulator call sleep shall detect and report the following exception (error) conditions.

None

3.2.41 stat

3.2.41.1 Inputs

The Emulator call stat shall take the following parameters.

pn: pointer to pathname string
ba: buffer address for file status information

3.2.41.2 Processing

The Emulator call stat shall return the file status information for the specified file. In the PDP-11/70 implementation of KSOS, the format of the information to be returned shall be consistent with the existing call except for such information which is meaningless under KSOS. These fields may be used for KSOS specific information. Certain values (e.g. link count, time of last access) may not be returned since they may be used as clandestine information channels. The access checks of the corresponding open call shall be satisfied in order for this call to succeed. If the file exists, but cannot be accessed due to mandatory security, integrity, or discretionary access reasons, a distinctive null status block shall be returned, however, the error condition shall not be raised.

3.2.41.3 Outputs

The Emulator call stat shall return the following values after execution.

fs: file status information
ec: error conditions

The Emulator call stat shall detect and report the following exception (error) conditions.

EX: file system required for pathname interpretation is not mounted
EX: file required for pathname interpretation is not a directory
EX: directory on path is unsearchable
EX: directory on path is at higher security level and is unreadable (returned as if directory was unsearchable)
EX: directory on path is at lower integrity level and is unreadable (returned as if directory was unsearchable)
EX: file does not exist

3.2.42 stime

The stime UNIX call has been replaced bv equivalent functionality in the Non-Kernel Security-Related Software.

3.2.43 stty

3.2.43.1 Inputs

The Emulator call stty shall take the following parameters.

fd: file descriptor
pp: pointer to parameter block
pb: device control parameter block

3.2.43.2 Processing

The Emulator call stty shall set the device dependent control parameters for the device referenced by the file descriptor. The format of the parameter block shall be the same as the existing call. Unlike Version 6.0 UNIX, the user must own the device for this call to be effective.

3.2.43.3 Outputs

The Emulator call stty shall return the following values after execution.

ec: error conditions

The Emulator call stty shall detect and report the following exception (error) conditions.

EX: invalid file descriptor
EX: device is not a character mode device
EX: not owner of the device
EX: parameters invalid (stty can be used for other devices which may include additional checking on the parameter block)

3.2.44 sync

3.2.44.1 Inputs

The Emulator call sync shall take the following parameters.

None

3.2.44.2 Processing

The Emulator call sync shall cause all file system information for the process family to be updated.

3.2.44.3 Outputs

The Emulator call sync shall return the following values after execution.

ec: error conditions

The Emulator call sync shall detect and report the following exception (error) conditions.

None.

3.2.45 time

3.2.45.1 Inputs

The Emulator call time shall take the following parameters.

None

3.2.45.2 Processing

The Emulator call time shall return the system time in seconds expired since January 1, 1970. The system time shall be expressed in Greenwich Mean Time (GMT).

3.2.45.3 Outputs

The Emulator call time shall return the following values after execution.

ti: time in GMT since January 1, 1970 (returned in R0 and R1)

The Emulator call time shall detect and report the following exception (error) conditions.

None

3.2.46 <u>times</u>

3.2.46.1 <u>Inputs</u>

The Emulator call times shall take the following parameters.

tb: pointer to data vector in user domain for execution time data

3.2.46.2 <u>Processing</u>

The times Emulator call shall return the time-accounting information for the currently executing process and its terminated children. The times shall be expressed in 1/10 seconds. The relevant information returned shall be the time spent by the process in user mode and Emulator mode execution, and the time spent by the terminated children in user mode and Emulator mode execution. The time elapsed in Kernel mode execution shall not be returned since this data opens a clandestine information channel through the Security Kernel.

3.2.46.3 <u>Outputs</u>

The Emulator call times shall return the following values after execution.

pu: process user mode execution time
ps: process Emulator mode execution time
cu: terminated child process user mode execution time
cs: terminated child process Emulator mode execution time
ec: error conditions

The Emulator call times shall detect and report the following exception (error) conditions.

EX: improper buffer address (not checked in Version 6.0 UNIX.)

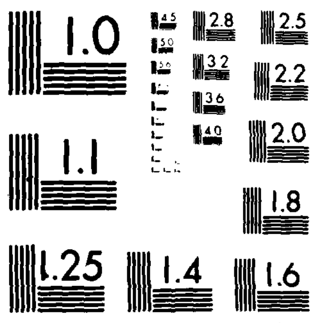
3.2.47 <u>umount</u>

The UNIX u(n)mount call shall be replaced by equivalent functionality in the Non-Kernel Security-Related Software.

AD-A111 578 FORD AEROSPACE AND COMMUNICATIONS CORP PALO ALTO CA W--ETC F/G 9/2
KSOS COMPUTER PROGRAM DEVELOPMENT SPECIFICATIONS (TYPE B-5). (K--ETC(U)
DEC 80 MDA903-77-C-0333
UNCLASSIFIED WDL-TR7932-REV-3 NL

3 OF 4
AD
A111578

111157



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

3.2.48 unlink

3.2.48.1 Inputs

The Emulator call unlink shall take the following parameters.

pn: pointer to file pathname string

3.2.48.2 Processing

The unlink Emulator call shall remove the directory entry for a file. If this was the last reference (link) to the file, the file contents shall be deleted. If the file is currently open, the actual removal shall be delayed until the file has been closed. The directory entry shall be removed immediately.

3.2.48.3 Outputs

The Emulator call unlink shall return the following values after execution.

ec: error conditions

The Emulator call unlink shall detect and report the following exception (error) conditions.

EX: file does not exist
EX: file system needed to interpret pathname is not mounted
EX: file on interpretation path is not a directory
EX: directory on name interpretation path has search access denied
EX: directory on interpretation path is at higher security level (returned as if directory had search access denied)
EX: directory on interpretation path is at lower integrity level (returned as if directory had search access denied)
EX: file to be removed is at lower security level (returned as if file had write access denied)
EX: file to be removed is at higher integrity level (returned as if file had write access denied)
EX: file to be removed has write permission denied
EX: parent directory of file has write access denied
EX: file to be removed is a directory and is not empty

3.2.49 wait

3.2.49.1 Inputs

The Emulator call wait shall take the following parameters.

None

3.2.49.2 Processing

The Emulator call wait shall cause the executing process to suspend execution until the termination of one of its children. If no children exist, the return shall be immediate with the appropriate error code.

In the PDP-11/70 implementation of KSOS, the normal return shall present the process ID of the terminated child to the parent in R0. The high order byte of R1 shall contain upon return, the exit status of the process (see exit above.) The low order byte of R1 shall contain the termination status of the child process. The following values shall be used to represent specific termination circumstances.

0: normal termination.
0177: stopped, but not terminated. (see ptrace below.)
0200: terminated, core image produced.

3.2.49.3 Outputs

The Emulator call wait shall return the following values after execution.

pi: process ID of terminated child (returned in R0)
rs: child return status from its exit (R1 high order byte)
ts: child termination status (R1 low order byte)

The Emulator call wait shall detect and report the following exception (error) conditions.

EX: no children belonging to calling process

3.2.50 write

3.2.50.1 Inputs

The Emulator call write shall take the following parameters.

fd: file descriptor (passed in register R0)
ba: buffer address of data vector to be written
nb: number of bytes to be written

3.2.50.2 Processing

The write call shall transfer data from the process user domain to the specified file, device or pipe. The number of bytes actually written shall be returned. This count shall be returned in register R0, in the case of the PDP-11/70 KSOS implementation. If an error has resulted, the actual byte count shall differ from the requested transfer count.

3.2.50.3 Outputs

The Emulator call write shall return the following values after execution.

bc: number of bytes actually written
ec: error conditions

The Emulator call write shall detect and report the following exception (error) conditions.

EX: invalid file descriptor
EX: file has not been opened for writing
EX: attempt to write a pipe without a reader at the other end
EX: physical input/output error
EX: illegal buffer address
EX: invalid requested transfer count

3.2.51 Directory Manager

The KSOS Emulator shall use the file subtype capability of the Security Kernel to implement UNIX compatible directories. For a detailed discussion of Kernel level file subtypes, the reader is referred to the KSOS Security Kernel Computer Program Development Specification (Type B5) [Kernel 78]. For this discussion, it is sufficient to state that the Kernel file subtype mechanism shall permit the overall set of file resources to be divided into subtype partitions. In order to access a file within a particular subtype partition, the process shall meet the discretionary access conditions predefined for files of that subtype, in addition to the usual mandatory security, integrity and discretionary access conditions. Access to files within a subtype shall be granted on the basis of the requested access mode (i.e., read, write, execute (or search)) and the access modes permitted to the requestor. The categories to which a user may belong shall be file subtype owner, user group, and the set of all users, conforming to the usual notion of UNIX discretionary access permissions.

The following paragraphs describe one possible implementation of the Directory Manager. The method presented below is a minimum privilege solution. It only requires the Directory Manager to temporarily execute in the discretionary access domain of the directory subtype owner and that the Directory Manager be privileged to use the K_link and K_unlink Kernel calls. These privileges are insufficient to allow the a Directory Manager at say, the SECRET level to create a directory entry in an UNCLASSIFIED directory. This would require that the Directory Manager also be privileged to violate the security *-property. Should the Government require that the Directory Manager be able to manipulate directories that are at security levels different than its (the Directory Manager's) current level, the Directory Manager shall be privileged to violate the security *-property, and shall control the bandwidth of the resulting storage channel. This privileged version of the Directory Manager shall also create the appropriate audit information whenever it actually violates the security *-property, and send the audit information via an IPC message to the Audit Capture Process [NKSR 78].

To support search-only directories, the directory manager must be able to read execute-only files. Therefore the directory manager shall be privileged to violate discretionary access in order to perform path name interpret functions on search only directories.

3.2.51.1 Inputs

KSOS Emulator files shall belong to one of two file subtypes, ordinary files and directory files. All user processes shall have read, write, and execute permission to ordinary files as defined by the subtype discretionary permissions. (Of course, before a user process can successfully access an ordinary file, the mandatory security and owner-permitted discretionary access conditions shall be fulfilled.) Directory files, however, shall have the following file subtype permissions:

a. file subtype owner: read, write, execute (search) enabled

b. subtype owner group: read, write, execute (search) enabled

c. all other users: read and execute (search) enabled

All KSOS users shall be able to read and search directory files. Only the owner of the subtype (the Directory Manager) shall be able to write or delete a directory file. Since directory files are still subject to mandatory security and discretionary access constraints, directory searches shall fail when the appropriate access conditions have not been satisfied.

When a user process intends to perform a directory operation that requires writing on a directory file, the process shall create a child process using the Kernel primitive K_spawn and then shall wait for the child process to return. The K_spawn call shall specify that the child process shall be the Directory Manager (i.e. that the child shall begin executing the image of the Directory Manager). Through the argument segment, the process shall specify the requested directory operation, the pathname of the directory, and other necessary parameters (i.e., file pathname, file SEID.)

3.2.51.2 Processing

When invoked, the Directory Manager process shall execute in the discretionary access domain of the directory subtype owner. The Directory Manager shall obtain explicit access capability to the file subtype, directory, by performing a Kernel level open on the subtype SEID. The Directory Manager shall change its access domain to the real group and user ID of the invoking process. The Directory Manager shall open the specified directory file at the Kernel level supplying the open object descriptor returned from the open on the directory subtype. The Directory Manager shall then perform any one of the following operations as requested by the invoking process.

a. create a new directory

b. make an entry in an existing directory

c. delete an entry from an existing directory

d. delete an existing directory

e. perform path name interpret functions.

Error conditions or return values shall be sent to the invoking process via IPC message. The Directory Manager process shall exit and cause the invoking process (waiting upon the death of the child) to resume execution.

3.2.51.3 Outputs

The Directory Manager shall detect and report the following exception conditions:

EX: directory file does not exist
EX: could not interpret directory pathname due to security or discretionary access violation
EX: attempt to delete non-empty directory
EX: file could not be found in directory

Exception conditions shall be returned to the requesting process via IPC message.

In addition, when the Directory Manager is forced to violate the security *-property, the Directory Manager shall create an audit record of the event, and send it via an IPC message to the Audit Capture Process.

3.2.52 Network Interface

The Emulator shall support the interface to a modern computer network. The protocol shall be the standard DOD Internet and Transmission Protocol [Postel 80a] [Postel 80b]. Table I shows the allocation of TCP functions to the three CPCI's which make up KSOS. In the remainder of this paragraph the TCP functions performed by the Emulator are discussed.

Table I

TCP Functional Allocation

	Security Kernel	UNIX Emulator	TCP Daemon
IMP Support	x		
Rendezvous			
Flow Control		x	
Data Stream Integrity:			
Checksum			x
Resequence		x	
Retransmission		x	
Acknowledge		x	
Duplicate Detect		x	
Routing			x
Connection Alloc.			x
Urgent Processing		x	
Formatting		x	
Security Level of Message			x

3.2.52.1 <u>Inputs</u>

The inputs to the network interface are the data stream(s) to and from the network, and control information. The control information shall normally be supplied as a side effect of the UNIX calls to the network (e.g., open, close, read, and write). If additional control information is required, the stty and gtty user calls may be utilized. The interpretation of the UNIX open call mode argument shall be modified when the "file" being opened is a network logical connection. If the mode is an even number above some defined minimum, it shall be interpreted as the address of a control block in the user address space that defines the parameters required for a general network connection to be established. The exact format of this control block is left to the discretion of the development contractor. Modes 0, 1, and 2 shall utilize a default set of parameters. Odd mode values shall be considered to be in error.

3.2.52.2 Processing

3.2.52.2.1 Initiation of Connections

The Emulator shall initiate a connection by passing a shared segment name to the Network Daemon (part of the Non-Kernel Security-Related Software CPCI) using the K_post Security Kernel call. Note that if the security levels of the user and the network are such that the requested communication cannot be established, the K_post call will fail. This shared segment shall be used for all subsequent communication between the Emulator and the Network Daemon. The initial contents of the shared segment shall be the parameters needed by the Network Daemon to allocate the connection. The Network Daemon and the Emulator shall use a protocol between themselves that assures that the shared segment will not become corrupted.

3.2.52.2.2 Flow Control

The Emulator shall provide flow control on each of the connections from the user process. In TCP this consists of selective advancing the the windows. The Emulator shall be designed to allow it to be tuned to improve overall performance. The tuning may be a dynamic adjustment of the window updating or may be a static (parameter) adjustment.

3.2.52.2.3 Data Stream Integrity

The Emulator shall provide most of the functionality in the area of data stream integrity. This shall include acknowledgement of received octets (bytes), retransmission of unacknowledged transmissions, detection of duplicate octets in the received stream, and resequencing the streams. The Network Daemon shall perform the checksum function on incoming and outbound data.

3.2.52.2.4 Urgent Processing

(The subject of urgent processing is currently under intense debate in the TCP community. The design requirements presented here should be reviewed in light of the latest conclusions of the community.)

The Emulator should notify the user that "urgent" data has not yet been received by him (the user). The user should upon receipt of such notification read the data stream up to the urgent pointer. The development contractor may suggest the manner in which the user is notified of the existence of unprocessed urgent data, and the way in which the start of the urgent data is communicated to the user. The preferred mechanism shall be via the signal mechanism.

3.2.52.2.5 Formatting

The Emulator shall break up the outbound data stream into letters and segments. The Emulator may allow the user to specify that the end-of-letter (EOL) bit is to be set at some point in the stream. In general, the formatting of the stream into segments and letters should be transparent to the user program.

3.2.52.3 Outputs

The output of the network interface function of the Emulator shall be the data streams to and from the user process and the network.

3.2.53 Special Requirements

The programming standards for the Emulator are discussed in the System Specification (Type A). Because the Emulator is not trusted by the Security Kernel, the emphasis on provability of the code is lessened.

If the Directory Manager is privileged (so it can create files in directories whose level is different than that of the process), it shall be subject to the same design and construction requirements as other privileged software.

3.2.53.1 Human Performance

This paragraph is not applicable to this specification.

3.2.53.2 Government Furnished Property List

The compiler for the language used to implement the Emulator may be furnished by the Government. The Government shall approve the implementation language to be used.

3.3 Adaptation

In general the Emulator should not require adaptation to the environment of a particular site. Because the Emulator is not trusted, it may be modified without affecting the security of KSOS. However, incorrect modification of the Emulator may adversely affect the UNIX interface. The parameters which control the sizes of buffer caches, and similar functions shall be adjustable by each site to tune the system.

4. QUALITY ASSURANCE PROVISIONS

4.1 Introduction

The underlying philosophy of KSOS quality assurance is to provide a convincing demonstration of the security and completeness of the system. Testing and quality assurance are a complement to the formal verification aspects of KSOS. All testing on KSOS shall be conducted at the development contractor's facility.

KSOS shall be subject to the technical reviews and audit procedures of MIL-STD-1521A, Appendices A-F. The following technical reviews and audits will be conducted:

a. System Requirements Review (Appendix A)

b. System Design Review (Appendix B)

c. Preliminary Design Review (Appendix C)

d. Critical Design Review (Appendix D)

e. Functional Configuration Audit (Appendix E)

f. Physical Configuration Audit (Appendix F)

The Critical Design Review shall include a review of any mathematical proofs showing that the design meets the requirements of the Government approved DoD security model.

The Functional Configuration Audit shall be the vehicle for the formal review of the design versus its specifications. The Physical Configuration Audit shall be the vehicle for the formal review of the "as built" computer programs versus both their design and their supporting documentation. In addition, the Physical Configuration Audit shall verify that the implementation requirements of the Type A System Specifications. have been met. Both audits shall include review of any relevant mathematical proofs of correctness.

4.1.1 Category I Testing

The development contractor shall establish test plans to demonstrate that all the requirements of Section 3 have been met. These tests shall be designed to exercise all the interface options for each function. To the extent feasible, the tests shall exercise all combinations of options. The tests shall include sequences intended to fail, such as attempts to violate the system's security rules or to use unacceptable combinations of options. Government approval of all test plans is required.

The general mechanism for assuring that the requirements of Section 3 have been met shall be a comparison of the system "state" before and after the function has been invoked. To ensure repeatability all testing shall be automated if possible. Such automation may include (but is not limited to) shell scripts, test programs, and terminal emulation by another computer system.

4.1.2 Category II Testing

In addition to the tests described above, the entire KSOS system shall be subject to on overall system test. These Category II tests shall be carried out in accordance with the provisions of the development contract, but should include at least one test with at least ten (10) terminals and a network connection simultaneously active. The Category II tests should exercise the entire system by simulating a typical user load. The Category II testing should be automated as much as possible through the use of shell scripts, test load programs, and terminal emulation by another computer system.

Table II - Quality Assurance

Assurance techniques:

NA	Not Applicable
V	Verification
F	Formal Specification
T	Testing
D	Demonstration
A	Analysis
I	Inspection

Applicable test types:

1	Module level testing
2	Integration testing
3	Acceptance testing

Requirement	Assurance Techniques							Test types		
	NA	V	F	T	D	A	I	1	2	3
3.1.1.				x	x		x	x	x	x
3.1.1.1.				x	x		x	x	x	x
3.1.1.2.1.				x	x		x	x	x	x
3.1.1.2.2.				x	x		x	x	x	x
3.1.1.2.3.				x	x		x	x	x	x
3.1.1.2.4.				x	x		x	x	x	x
3.1.1.2.5.				x	x		x	x	x	x
3.2.1. - 3.2.50.				x	x	x	x	x	x	x
3.2.51.			*	x	x	x	x	x	x	x
3.2.52.			*	x	x	x	x	x	x	x
3.2.53.	x									
3.2.53.1.	x									
3.2.53.2.	x									
3.3.						x	x			

* Security related functions to be formally specified.

5. PREPARATION FOR DELIVERY

This section is not applicable to this specification.

6. NOTES

This section is not applicable to this specification.

10. APPENDIX

WDL-TR7934
Revision 3
December 1980

SECURE MINICOMPUTER OPERATING SYSTEM (KSOS)

NON-KERNEL SECURITY RELATED SOFTWARE COMPUTER PROGRAM DEVELOPMENT SPECIFICATIONS (TYPE B5)

Department of Defense Kernelized Secure Operating System

Contract MDA 903-77-C-0333
CDRL 0002AH

Prepared for:

Defense Supply Service - Washington
Room 1D245, The Pentagon
Washington, DC 20310

Ford Aerospace &
Communications Corporation
Western Development
Laboratories Division

3939 Fabian Way
Palo Alto, California 94303

NOTICE

The Department of Defense Kernelized Secure Operating System (KSOS) is being produced under contract to the U.S. Government. KSOS is intended to be compatible with the Western Electric Company's UNIX*tm Operating System (a proprietary product). KSOS is not part of the UNIX license software and use of KSOS is independent of any UNIX license agreement. Use of KSOS does not authorize use of UNIX in the absence of an appropriate licensing agreement.

This document is a major revision of an earlier draft dated July 1978. That and other previous editions are obsolete and should not be used.

UNIX and PWB/UNIX are trade/service marks of the Bell System.

DEC and PDP are registered trademarks of the Digital Equipment Corporation, Maynard MA.

TABLE OF CONTENTS

Kernelized Secure Operating System (KSOS): Non-Kernel Security Related Software Computer Program Development Specification (Type B5)

1. SCOPE

1.1 Identification

This specification establishes the performance, design, development and test requirements for the Non-Kernel Security Related Software Computer Program Configuration Item (CPCI) which is part of the Kernelized Secure Operating System (referred to as "KSOS"). KSOS provides a provably secure, resource-sharing operating system compatible with the standard user environment provided by UNIX, as described by [Thompson 75] and [Ritchie 74]. The Non-Kernel Security Related Software provides those functions which are needed by the system for continued operation and maintenance, and which are not part of the Security Kernel CPCI.

1.2 Functional Summary

The Non-Kernel Security Related Software is a collection of autonomous subsystems; some of which execute with extraordinary privilege (i.e. more privilege than is afforded to user programs) to provide essential services to the system. Such services include, but are not limited to, the following:

- Secure User Services: those services invoked by users which require a trusted path from the user to the service (e.g., the login process).
- System Operation Services; those functions essential to the operation of a KSOS system (e.g., the Process Bootstrapper).
- System Maintenance Services; those functions needed for continued operation and maintenance of a KSOS system, such as dump and restore of file systems.
- System Administrator Services; those functions needed to support the administrative operation of the system, such as the adding and deleting of users.

The remainder of this specification is organized as follows:

- Section 2 contains the references
- Section 3 contains the requirements for the CPCI
- Section 4 contains the quality assurance provisions for the CPCI
- Sections 5 and 6 are not applicable and are, therefore not present.

1.3 Terminology

The language used throughout this specification attempts to conform to the guidelines of Section 3.2.3 of MIL-STD-490. In particular, the word "shall" means that the specification expresses a provision that is binding. The words "should" and "may" mean that the specification expresses a provision which is non-mandatory. The word "will" is used to express a declaration of purpose on the part of the Government.

KSOS shall support the mandatory DoD security policy of DoD Directive 5200.1-R which is embodied in a Government approved mathematical model. [Verif 78] Proofs of the system's security properties shall be in terms of this model. KSOS shall provide a minimum of eight (8) hierarchical security classifications categories (or simply security categories) and a minimum of thirty-two (32) mutually independent security compartments. The security categories shall be such that

UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP SECRET

Where "<" is defined in accordance with the requirements of DoD 5200.1-R. One security compartment shall be reserved for read protection of system data bases and programs. This compartment shall be called the "system" compartment.

KSOS shall provide for Kernel-enforced integrity. Integrity is defined as the formal mathematical dual of security [Biba 75]. At least four (4) hierarchical integrity classifications categories (or simply integrity categories) shall be provided in KSOS. The integrity categories include at least the following three classification categories:

USER < OPERATOR < ADMINISTRATOR

Although there is no immediate requirement for integrity compartments, the development contractor should include provisions to ease their later inclusion.

For the remainder of this specification, the term "security level" means the combination of a security category and a set of security compartments (which may be null). The term "integrity level" means the combination of an integrity category, and a (presently always null) set of integrity compartments. The term "level" (or access level) means the security and integrity level; i.e., the combination of a security category, a set of security compartments (which may be null), an integrity category, and a (presently always null) set of integrity compartments. The "access matrix" defines, for a particular user or group, the combination of the maximum security level permitted for that user or group, all security compartments to which that user or group has access, the maximum integrity level for that user or group, and all integrity compartments (presently always null) to which that user or group has access.

In a secure system, it is necessary to have processes external to the kernel which perform security critical functions. These processes have come to be called "trusted processes". In KSOS, there is a refinement in this terminology. As Figure 1-1 shows there are subdivisions of the software inside the security perimeter. Some of the processes are "privileged", in other words, they may violate one or more of the security rules enforced by the Kernel. These processes must be designed and implemented with the same care and thoroughness as the Kernel itself. Section 3.1.1.2.4.4 presents the privileges which may be afforded processes.

A second subdivision has been termed "responsible" processes. These processes are not privileged to violate any of the Kernel's rules, but which are, nonetheless, important to the overall security of the system. Examples of responsible processes include the Secure Mail facility and the processes which manipulate the data bases describing the users' security and integrity permissions (the "access matrix"). These processes must also be carefully designed and implemented, but do not require formal specification or verification. Because they do perform security related functions, they cannot include untrusted components (e.g. the UNIX Emulator).

The third subdivision is the "encapsulated utilities" (EU in the figure). These are self-contained functions to which the user simply supplies parameters. The encapsulated utilities are not privileged to violate Kernel rules and do not affect the security of users. The chief function in the encapsulated utilities is system generation.

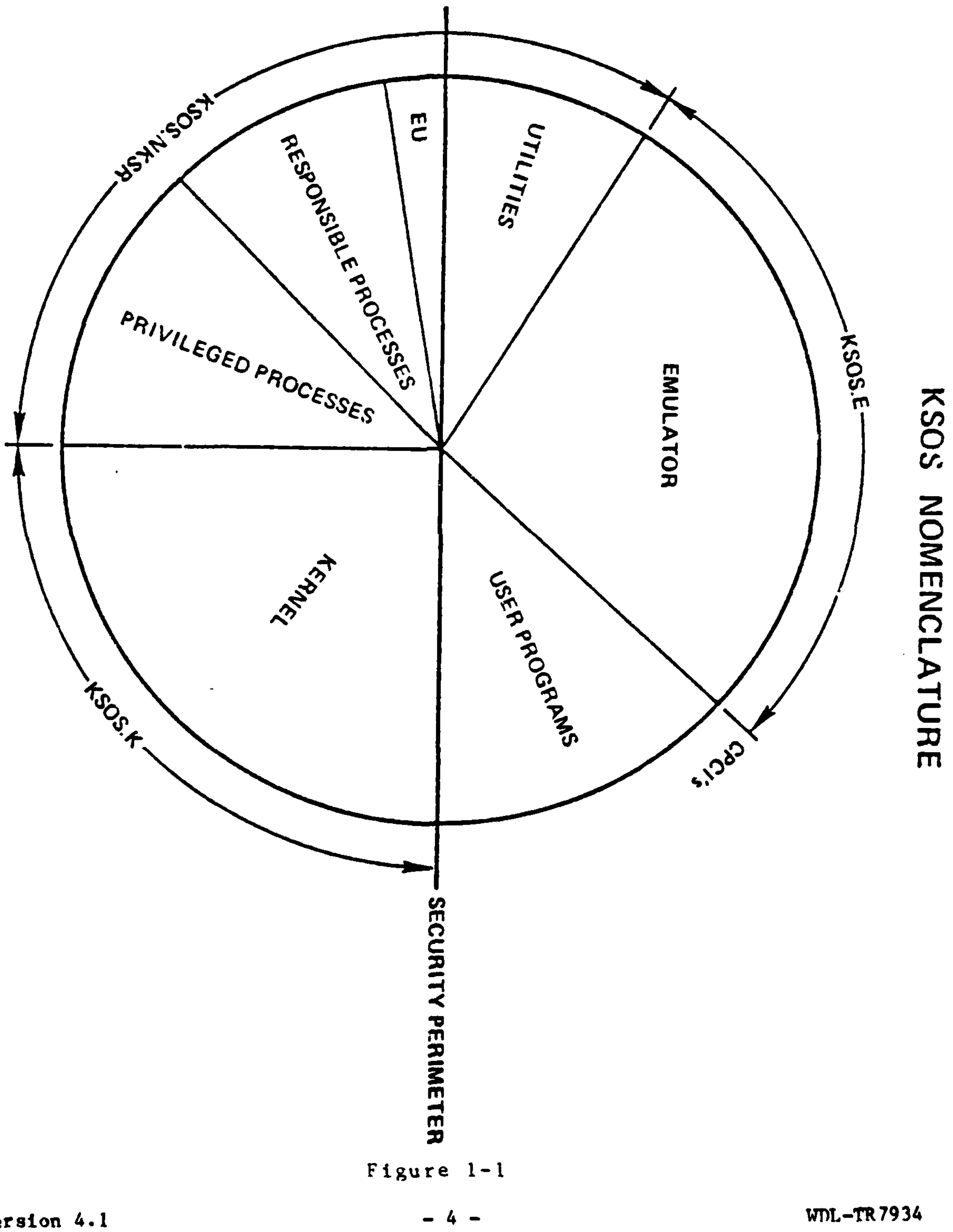

KSOS NOMENCLATURE
KSOS.NKSR
KSOS.E
KSOS.K
CPCI's
EU
UTILITIES
RESPONSIBLE PROCESSES
PRIVILEGED PROCESSES
EMULATOR
KERNEL
USER PROGRAMS
SECURITY PERIMETER

Figure 1-1

2. APPLICABLE DOCUMENTS

The following documents, of exact issue shown, form a part of this specification to the extent specified herein. In the event of a conflict between the referenced documents and the contents of this specification, this specification shall be considered a superseding requirement. In the text, references to these documents are in the form [Name date], e.g. [Biba 75].

2.1 Government Documents

2.1.1 Directives, Manuals and Standards

a. DoD 5200.1-R Information Security Program Regulation

b. DoD 5200.28 Security Requirements for Data Processing (ADP)

c. DoD 5200.28-M ADP Security Manual

d. MIL-STD-483 Configuration Management

e. MIL-STD-490 Specification Practices

f. MIL-STD-1521A Technical Reviews and Audits

2.1.2 Reports

a. [A-Specs 78] "KSOS System Specification (Type A)", WDL-TR7808 Revision 1, Ford Aerospace and Communications Corporation, Palo Alto, CA (July 1978).

b. [Bell and LaPadula 73] Bell, D.E. and LaPadula, L.J., "Secure Computer Systems", ESD-TR-73-278, Volume I-III, MITRE Corporation, Bedford, MA (November 1973 - June 1974).

c. [Biba 75] Biba, K.J., "Integrity Considerations for Secure Computer Systems", MTR-3153, MITRE Corporation, Bedford, MA (June 1975).

d. [Emulator 78] "KSOS Computer Program Development Specification (Type B5): UNIX Emulator", WDL-TR7933, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

e. [Kernel 78] "KSOS Computer Program Development Specification (Type B5): Security Kernel", WDL-TR7932, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

f. [Verif 78] "KSOS Verification Plan", WDL-TR7809, Ford Aerospace and Communications Corporation, Palo Alto, CA (March 1978).

g. [Walter et al. 74] Walter, K.G. et al., "Primitive Models for Computer Security", ESD-TR-74-117, Case Western Reserve University, Cleveland, OH (January 1974).

2.2 Non-Government Documents

a. [BBN 75] "Interface Message Processor, Specifications for the Interconnection of a Host and an IMP", Report 1822, Bolt Beranek and Newman Inc., Cambridge, MA, (December 1975).

b. [Holmgren et al. 77] Holmgren, S.F., Healy, D.C., Jones, P.B. and Kasprzycki, E., "Illinois Inter-Process Communication Facility for UNIX", CAC Technical Memorandum 84, CCTC-WAD Document 7507, Center for Advanced Computation, University of Illinois at Urbana-Champaign, Urbana, IL (April 1977).

c. [Lampson 73] Lampson, B., "A Note on the Confinement Problem", CACM, Volume 16, Number 10, pp 613 - 615 (October 73).

d. [Lipner 75] Lipner, S.B., "A Comment on the Confinement Problem", Proc. Fifth Symposium on Operating Systems Principles, ACM SIGOPS Review, Volume 9, Number 5, pp 192 - 196 (19-21 November 1975).

e. [Nemeth et al. 77] Nemeth, A.G., Sunshine, C., Zucker, S. and Tepper, "Progress Towards a DeFacto DoD UNIX Inter-Process Communication Standard", Working Paper, (May 1977). (Available from the authors.)

f. [Parnas 72] Parnas, D.L., "A Technique for Software Module Specification with Examples", CACM, Volume 15, Number 5, pp 330 - 336 (May 1972).

g. [Postel 78a] Postel, J.B., "Internetwork Protocol Specification", Version 4 , Information Sciences Institute, University of Southern California, Marina del Rey, CA (February 1979).

h. [Postel 78b] Postel, J.B., "Specification of Internetwork Transmission Control Protocol - TCP", Version 4 (draft), Information Sciences Institute, University of Southern California, Marina del Rey, CA (February 1979).

i. [Roubine and Robinson 77] Roubine,O., L.Robinson, Special Reference Manual, 3rd ed., Technical Report CSG-45, SRI International, Menlo Park, CA (January 1977).

j. [Sunshine 77] Sunshine, C., "Interprocess Communication Extensions for the UNIX Operating System: I Design Considerations", R-2064/1-AF, RAND Corporation,

k. [Zucker 77] Zucker, S., "Interprocess Communication Extensions for the UNIX Operating System: II Implementation", R-2064/2-AF, RAND Corporation, Santa Monica, CA (June 1977).

2.3 Other References Not Part of This Specification

a. [Thompson 75] Thompson, K. and Ritchie, D.M., "UNIX Programmer's Manual", Sixth Edition, Western Electric Corporation, Greensboro, NC (May 1975).

b. [Ritchie 74] Ritchie, D.M. and Thompson, K., "The UNIX Timesharing System", CACM, Volume 17, Number 5, pp 365 - 375 (May 1974).

c. [UNIX 75] "UNIX Program Listings", Version 6.0, Western Electric Corporation, Greensboro, NC (May 1975).

3. REQUIREMENTS

The Non-Kernel Security Related Software shall provide those security relevant functions which are not allocated to the Security Kernel CPCI. Certain functions of the CPCI may be privileged to violate one or more of the security rules of the Security Kernel in providing overall system security. These privileged functions and the rules which they violate are identified in this section. The privileged functions must be implemented with the same precision and thoroughness as the Security Kernel. The System Specifications (Type A) [A-Specs 78] for KSOS present the requirements and constraints for the design and the implementation. Non-Kernel Security Related Software shall operate correctly without modification on all hardware configurations defined in the System Specification for KSOS.

Several of the functions described in this section require constraints on their user; e.g., to use some of these programs, the user must be at a certain integrity level or higher. These programs must be run in the "SYSTEM" compartment which is reserved for such programs and data bases. These steps assure that the executable images and data bases remain untampered with and that the programs cannot be invoked successfully by unauthorized users. For compactness, some of the functions are described as being available only to the System Operator, which means a user running at OPERATOR level or higher level and in the "system" security compartment.

3.1 Computer Program Definition

3.1.1 Interface Requirements

The Non-Kernel Security Related Software may utilize the Security Kernel interface and, in some cases, may use the UNIX interface provided by the UNIX Emulator. These two interfaces are specified in the companion Type B5 specifications for these CPCI's.

3.1.1.1 Interface Block Diagram

Figure 3-1 shows the relationship of the functional components of KSOS.

```
             ._______________________________________________.
             | User Programs| untrusted     .                 |
USER MODE    | (may include | NKSR          .                 |
             | Kernel calls)|               .                 |
             |______________|_______________..................|
SUPERVISOR   |                              |                 |
MODE         |         UNIX EMULATOR        |trusted NKSR     |
             |______________________________|_________________|
KERNEL MODE  |        SECURITY KERNEL                         |
             |________________________________________________|
```

(NKSR: Non-Kernel Security Related Software)

Figure 1-2. Functional Components of KSOS

The call from domains of lesser privilege to those of greater privilege shall be mechanized by means of an inter-domain call. In the PDP-11/70 implementation the TRAP and IOT instructions shall be used by user programs to call the UNIX Emulator. The EMT instruction shall be used by user programs and the Emulator to call upon the Kernel.

It should be noted that no privileged functions may use the Emulator, since the Emulator is not trusted. Those functions in the Non-Kernel Security Related Software which are not privileged (e.g., the Secure Mail spooler) may use the Emulator.

3.1.1.2 Detailed Interface Definition

3.1.1.2.1 Secure User Services

The Secure User Services class of Non-Kernel Security Related Software consists of those functions which require a secure path to and from the user, such as the login process. In general, the functions which require such a secure path are those in which the user may be supplying raw passwords or the names of objects to be manipulated in a trusted way such as changing the security of an object.

3.1.1.2.1.1 Secure Initiator

The Secure Initiator shall be invoked by the Initial Process when the system is booted. The Secure Initiator shall spawn off the Audit Capture Process (ACP), initialize various objects, and executed a file of commands similar to the rc files on UNIX. The Initiator shall also spawn one Secure Server process (SSP) for each configured terminal in the system. A data base containing a list of the configured terminals paired with the appropriate Secure Servers id created by the Secure Initiator.

3.1.1.2.1.2 Secure Server

One Secure Server process shall be present for each configured terminal in a KSOS system. The Secure Server shall receive commands from the user at the terminal via the secure path, and shall spawn processes on behalf of the user to carry out these commands.

Each terminal in the system shall logically consist of a set of mutually exclusive logical data-paths. Only one path shall be active at any one time. All terminal input shall be directed to the currently active path, and an attempt to read or write on a path other than the active path shall cause the requesting process to become blocked. One path shall be denoted as the secure path, and shall be used exclusively by the Secure Server. The secure attention character shall cause the kernel to select the secure path as the active path. Selection of an alternative terminal path shall be performed by a SET_PATH operation of the K_device_function call. This shall be a privileged operation whose use is restricted to the Secure Server.

On receipt of a secure attention character, the Secure Server shall prompt for input from the terminal. The Secure Server shall interpret the input as a request for a Secure User Service. If the user is authorized to use the service, the service shall be performed, either by the Server, or by spawning a process to perform the service on behalf of the user.

Some Secure User Services shall be capable of changing their security and integrity levels, and real owner (user ID and group ID) while running. This capability provides the mechanism necessary to perform in a controlled manner those user requests which either change the user's level or which require actions to be performed at another level. Because the Server has access to the user's access matrix it is able to verify the user's access level restrictions before performing any requested functions. Additionally, certain Secure processes can access restricted databases which system users are unable to access. These processes allow for the updating of these databases in a controlled and audited manner.

The Secure User Services under the direction of the Secure Server shall provide for authentication of users, control of user access levels, password modification, file access modification, and the spawning of level preserving copy and print functions.

3.1.1.2.1.3 Access Authentication Functions

A set of the Secure User Services known as the Access Authentication Functions shall service all user requests for login, changing group, changing user access level (access level includes security and integrity level), and logout.

3.1.1.2.1.3.1 Processing Operator Requests

The login and logout functions shall accept the following commands from the System Operator to control system access.

a. Logout a Terminal - Requests to logout a terminal shall be handled in the same manner as if requested by the user. Whether or not the terminal was logged in, the terminal mode shall be set to the values listed for it in the Terminal Profile Data Base. This provides a mechanism to regain control of a terminal in an incorrect mode.

b. Logout All Terminals - A check shall be made of all terminals configured in the Terminal Profile Data Base and a logout shall be performed on all terminals except the terminal designated as the operator's console.

c. Lock a Terminal - This command shall prevent logins on the designated terminal. This function will not force a logout on a terminal that is logged in when the command is issued, although the System Operator may do so with another command if desired.

d. Lock All Terminals - This command shall prevent login on all terminals except the operator's console. However, terminals already logged in will not be logged out.

e. Unlock a Terminal - This command shall allow logins on the designated terminal.

f. Unlock All Terminals - This command shall allow logins on all terminals.

g. Set System Security Level - This command shall set the maximum security level permitted on the system and this maximum level shall be checked when a user makes a change level or login request. It shall not affect the level of any user logged in when the command is issued. If the system level is being lowered, the System Operator may wish to logout users above that level, or he (she) may simply notify them to change their levels or logout as soon as possible, forcing a logout only if they are unresponsive. It shall not be possible to raise the system level above the system maximum level defined in the System Database.

3.1.1.2.1.4 <u>Password Modification Function</u>

The Password Modification Function shall allow a user to change his login password. It shall also allow the user designated as the Group Administrator in the Group Access Authentication Database to change the group password for that group. The Password Modifier shall not allow a user to change the password of another user. If a user is unable to login because of a bad password he must request the System Administrator to change it to a temporary password, allowing him to login and enter a new password.

3.1.1.2.1.5 <u>File Access Modification Function</u>

The File Access Modification Function shall allow a user to modify access levels including security, integrity, and discretionary access levels on files which he owns. It shall not allow him to grant any special privileges to his files.

3.1.1.2.1.6 <u>Level Preserving Copy Function</u>

The Secure User Services shall provide facilities to copy files while maintaining their security and integrity levels without the requirement of the user being at the same level as the file. Since objects created by processes are always created at the access level at which the process is running, normal copies would often result in the unnecessary upgrading of files. All security checks shall remain in effect for this type of copy. The test to determine if a Level Preserving Copy will be successful is to determine if the user would be able to change his level to the level of the file and then attempt do a normal copy.

If the above test is successful the Secure Server shall spawn a standard copy at the level of the file to be copied and with the requesters owner and group. The copy may still fail if the copy process is unable to open the input file or create the output file because discretionary access or other constraints.

3.1.1.2.1.7 <u>Level Preserving Print Function</u>

The Level Preserving Print Function operates in the same manner as the Level Preserving Copy Function except that the output is spooled for printing on the line printer.

3.1.1.2.2 System Operation Services

The System Operation Services class of Non-Kernel Security Related Software shall consist of those functions which are essential to the continuing operation of the system. These functions shall have separate interfaces and, in general, are independent of each other.

3.1.1.2.2.1 Network Daemon

The Network Daemon shall multiplex and demultiplex the data streams to and from the computer network. The Network Daemon shall be privileged to allow it to handle a network that can support communications at varying security levels. The Network Daemon shall have three main interfaces:

a. the Interface Message Processor (IMP) (or equivalent) interface

b. the Daemon Control Interface

c. the Network User Interface

3.1.1.2.2.1.1 IMP Interface

The Network Daemon shall communicate with the network via the IMP interface that is provided by the Security Kernel. This IMP interface shall appear to the Network Daemon as a device. For functions not common to all devices, the K_device_function Kernel call shall be employed. The main part of the IMP interface of the Network Daemon shall be the mechanization of the Host-to-Imp Protocol as defined by [BBN 75] (or other equivalent interfaces as specified by the Government in the development contract) and the mechanization of the Internetwork Datagram Layer [Postel 78a] of the Transmission Control Protocol (TCP).

3.1.1.2.2.1.2 Daemon Control Interface

The Daemon Control Interface shall be the mechanism by which the System Operator (or other authorized user) controls the state of the Network Daemon, and thereby the state of all network connections. The Daemon Control Interface shall be mechanized by means of inter-process communication from another process running at the OPERATOR or higher integerity level. The following Daemon Control commands shall be provided (the development contractor may extend or modify this list to suit the needs of a particular Network Daemon implementation):

a. Reset Daemon - shall return the Network Daemon to its initial state. This function shall include destroying all logical connections, transmitting any required resetting messages to remote hosts, and similarly informing the local end of all network connections. It should be used only in cases where the Network Daemon has become hopelessly confused.

b. Reset Host - shall return the state of a particular remote host to its initial state, destroying all open connections to that host.

c. Abort Connection - shall destroy a particular connection. Both the local and remote ends of the connection shall be informed of the aborting of the connection via the normal Network Daemon status returns.

d. Dump Status - shall cause the Network Daemon to dump its internal status information to a file. Because the status of the Daemon may involve connections of differing security levels, the classification of the status file shall be system high.

3.1.1.2.2.1.3 User Interface

The User Interface to the Network Daemon shall be via inter-process communication and shared segments. The bulk of the TCP functions shall be performed in the UNIX Emulator [Emulator 78]. Thus, a UNIX user-mode program would not communicate directly with the Network Daemon. Rather, the system calls from the user-mode program shall be interpreted by the UNIX Emulator and transformed into inter-process communication messages to and from the Network Daemon. The user shall initiate network communications using the UNIX "open" system call. To provide the additional information needed to open a network connection, the "mode" argument shall be subject to an extended interpretation. If the "mode" argument is even and greater than some defined constant and the "open" is for a network connection, then the argument shall be interpreted as the address of a control block. The exact format of the control block is left to the discretion of the development contractor.

The UNIX Emulator shall interface with the Network Daemon in the following manner. The opening of connections shall be accomplished by the transmission of IPCs to the daemon. The IPCs shall specify the identity of a sharable segment for subsequent data transfers between the emulator and the daemon. Completion of a connection open request by the daemon shall be notified to the emulator by an IPC message containing status information. Reading to and writing from the network shall be accomplished by passing the data via the shared segment, with control information for each transfer being passed in IPC messages. Multiple connections from a particular user may be multiplexed over the same shared segment. The UNIX Emulator and the Network Daemon shall obey a protocol that preserves the integrity of the shared segment (for example, the protocol could ensure that they both do not try to write the segment simultaneously, although other less restrictive protocol possibilities exist).

To facilitate the construction of network applications which do not include the UNIX Emulator, the TCP handling in the Emulator shall be designed to be as separate as possible from the remainder of the Emulator.

3.1.1.2.2.2 Line Printer Spooling

KSOS shall provide a secure mechanism for spooling output to the printer. The mechanism shall be secure in the sense that users cannot alter, view, or delete the spooled output of other users. In addition, the appropriate headers and footers shall be placed on the printed output. Finally, a record of the creation of classified printer output shall be generated. There are two interfaces to the Spooler:

a. The User Interface - similar to the interface to the present UNIX spooler (/bin/opr and /lib/lpr).

b. The Line Printer Daemon Control Interface - allows a System Operator (or other privileged user) to control the Daemon (de-spooler).

3.1.1.2.2.2.1 User Interface

The interface to the Line Printer Spooler shall allow the user to:

a. Print multiple copies of a file (or files)

b. Allow the user to specify an alternate heading on the burst pages
The User Interface shall be implemented as arguments to the Spooler program, using the standard UNIX argument passing conventions.

3.1.1.2.2.2.2 Line Printer Daemon Control Interface

The Line Printer Daemon Control Interface shall be via inter-process communication and shall allow the System Operator (or other privileged users) to:

a. Abort the printing of a file

b. Backup and reprint a page (for paper changes)

3.1.1.2.2.3 Assign/Deassign

The Assign/Deassign functions shall allow users to become owners of certain physical devices without having the privilege to do general ownership changing. Assign shall only allow the assignment of devices from a pool of free devices. It shall not be possible to assign a device that is presently in use or which is reserved for the system or another user.

Assign and Deassign shall be spawned through the Secure Server mechanism. The user's interface to Assign shall consist of specifying the device(s) to be assigned . The user interface to Deassign shall consist of specifying the device(s) to be deassigned. The System Operator shall be able to force a device to be deassigned.

The ability to assign ownership of a physical device such as a tape or disk drive to a user shall allow the user to use that device to access his media without risk of unauthorized access by other users. The user should assign a device before physically placing his media in it. This mechanism shall protect media from unauthorized access between the time the media is physically mounted and the time the user process open the device.

3.1.1.2.2.4 Mount/Unmount

Mount and Unmount shall be spawned through the Secure Server mechanism. The user shall supply the name of the file system to be mounted, the drive on which the file system resides, the pathname of a presently accessible entry where the file system is to be mounted (normally, the file would be of the directory subtype), and an optional argument indicating that the file system is to be mounted read only.

The user shall have discretionary access to the extent (i.e., if the extent is to be mounted read/write, the user must have read/write access; if the extent is to be mounted read only, the user must have read access) and write discretionary access to the entry where the file system is to be mounted to be able to successfully request that it be logically mounted.

The interface to the Unmount Process shall also be similar to that in the current UNIX system. The user shall supply the name of the file system to be unmounted.

3.1.1.2.2.5 System Startup/Shutdown

The user interface to the System Startup Process shall consist of supplying the name and location (i.e. extents) of the Security Kernel and Initial Process images to be loaded. This interface is provided by a small bootstrap program which is loaded using the ROM bootstrap or other mechanisms. The Initial Process shall then ask the user for the location of the root file system and for the time.

The System Shutdown Process shall be invoked by logging in as a special user whose default "shell" is a system shutdown program. This login shall be permitted only from the system console, and shall be protected by the usual password protection mechanisms.

3.1.1.2.2.6 Secure Mail

The user interface to the Secure Mail function shall consist of two programs which allow for the sending and receiving of mail. The basic interface shall be similar to the interface provided by the current UNIX mail program. However, the receive mail portion shall only deliver mail that exists at the security level of the Secure Mail process. The user interface shall allow the user to send a file or input from his terminal to one or more other users identified by their login identifiers. Mail may be sent by users running at any security level and will be protected at that level. The receive part of the Secure Mail function shall allow a user to read all mail sent to him that is at his current security level. To read mail at another security level, the user shall be required to change his security level.

3.1.1.2.3 System Maintenance Services

The System Maintenance functions shall be spawned through the Secure Server mechanism. Input parameters and output are defined in the Detailed Functional Requirements below.

3.1.1.2.4 System Administrative Services

The System Administrative Services class of Non-Kernel Security Related Software shall consist of the functions to support the administrative operation of the system, such as adding and deleting users.

3.1.1.2.4.1 User Control Editor

The User Control Editor shall be used by the System Administrator to control user access to the system. Specifically, the process shall be used by the System Administrator to add and delete users and groups from the system, to change user and group access level capabilities, to change a user's initial login group, and to set the identification of the Group Administrator for a group. These changes shall be exercised by updating the User Access Authentication Data Base and the Group Access Authentication Data Base. Since all decisions about access in KSOS are made on the basis of information contained in these data bases, the interface to this function shall be via the secure terminal path; i.e., the function shall be spawned off by the Secure Server. Alteration of these data bases shall not affect currently logged in users until they request some service for which one of the data bases would be checked. It is recommended that the System Administrator request (or force) a logout of presently active users affected by changes to these data bases.

3.1.1.2.4.2 Immigration Officer Process

The function of the Immigration Officer shall be to check the acceptability of "foreign" file systems. The immigration officer process shall take the name of the file system to be immigrated. It will restore (see File System Restore CPC 3.2.2) the file system from tape and checks its acceptability. Only after the Immigration Officer has approved the file system shall it be acceptable for mounting.

3.1.1.2.4.3 Audit Capture Process

The Audit Capture Process shall receive, via either the signal or the inter-process communication mechanisms, raw audit information from the Kernel and NKSR processes. It shall write this information to an audit file. The process shall be able to select the types of information to be kept and may add additional information such as a time stamp to the information before writing. Multiple audit files shall be used to save the audit information. The process shall accept a command from the System Operator to switch audit files.

3.1.1.2.4.4 Privilege Control Process

The Privilege Control Process shall provide the Security Officer with the ability to grant and revoke special access privileges for process image files. Privileges shall be given to processes by changing elements of the type independent information for the file from which the process is invoked. The privileges of presently active processes shall not be changed by the Privilege Control Process. The following privileges shall be capable of being granted or revoked:

a. The ability to violate the simple security property. (This privilege is not resently used. It is included for completeness.)

b. The ability to violate the simple integrity property.

c. The ability to violate the *-property for security.

d. The ability to violate the *-property for integrity.

e. The ability to violate discretionary access.

f. The ability to use the K_secure_terminal_lock primitive.

g. The ability to use the K_release_openers primtive.

h. The ability to use the K_link and K_unlink primitives.

i. The ability to use the K_mount and K_unmount [illegible]itives.

j. The ability to change the following type independent information about an object:

   1. The security level of the object.

   2. The integrity level of the object.

   3. The owner of the object.

k. The ability to set type dependent information (status) about an object; i.e., the ability to open an object in status update mode.

l. The ability to save the process image in the swapping area; i.e., the ability to set the "sticky bit" (ISVTXT).

m. The ability to lock a segment in core (not swappable).

n. The ability to grant and revoke privileges.

3.2 Detailed Functional Requirements

3.2.1 Secure User Services

A set of functions shall be provided which communicate with the user terminal through a secure communications path. These functions shall be known as Secure User Services.

3.2.1.1 Secure Initiator

3.2.1.1.1 Users and Privileges

The Secure Initiator shall be invoked by the Kernel as part of the System by the Initial Process as part of system initialization.

3.2.1.1.2 Inputs

The Secure Initiator will receive, from the Initial Process, the seids for the user and NKSR process bootstrapper segments.

3.2.1.1.3 Processing

The Secure Initiator shall spawn off the Audit Capture Process and shall make UNIX directory entries for the Process Bootstrapper segments. It shall then initialize the mount table and the access levels of various subtypes and devices. At this point, the Secure Initiator shall execute a shell file of commands similar to the rc files on UNIX. Finally, the Secure Initiator shall set the level of each configured terminal and spawn off a Secure Server for each such terminal.

The Secure Initiator shall also set up the terminal data base which shall contain a list of terminals coupled with the Secure Server running on that terminal. This data base shall then be used by the Secure Servers.

3.2.1.1.4 Outputs

The Secure Initiator shall pass to each Secure Server the identification of the terminal on which it runs.

3.2.1.2 Secure Server

3.2.1.2.1 Users and Privileges

A separate Secure Server shall be a spawned to service each terminal.

This Server shall be privileged to use the K_secure_terminal_lock Kernel call, to change certain type independent information about an object, and to change process type independent information.

3.2.1.2.2 Inputs

The process shall receive input from the Secure Initiator in the form of an argument segment, and from the user terminal via the secure terminal path.

Inputs from the user shall consist of the name of the service to be performed and an arguments which may be needed by that service.

3.2.1.2.3 Processing

The Secure Server shall prompt the user and read from the secure terminal path to determine type of service required. If the Server is unable to determine the type of service required, it shall display a rejection request on the user's terminal and thaw the terminal. In the case of operator requests, the Server shall determine that the user is currently running at OPERATOR integrity level or higher before an the function is performed. In the case of administrator requests, the Server shall determine that the user is currently running at ADMINISTRATOR integrity level before an the function is performed.

The Secure Server shall process user requests using a combination of spawned processes as needed to service user requests. Processing required for each type of service is described in the processing section for that service.

3.2.1.2.4 Outputs

The Secure Server shall communicate to the terminal user via the secure terminal path. Upon receiving an IPC signal from the Initiator the Secure Server shall display the user's current security level as a character string sent out on the secure path. Process which are spawned by the Secure Server shall receive their arguments via an argument segment.

Outputs required for each type of service are listed in the outputs section for that service.

3.2.1.3 Access Authentication Functions

The Access Authentication Functions shall process requests for login, change group, setting terminal access level, and logout. These functions shall be internal to the Secure Server process.

3.2.1.3.1 Inputs

Valid user names and identifiers, encrypted passwords, and the user access matrix shall be available from the User Access Authentication Data Base.

Valid group names and identifiers, encrypted passwords, and the group access matrix shall be available from the Group Access Authentication Data Base.

Profile information for each terminal shall be available from the Terminal Profile Data Base. The user's name, unencrypted password, and group name shall be obtained via the secure terminal path.

3.2.1.3.2 Processing

The Access Authentication Functions shall process user requests for logging in, logging out, changing group ,and changing access level. Additionally, they shall process certain requests from the System Operator which control login/logout.

3.2.1.3.2.1 Login Processing

After receiving a login request the current state of the terminal shall be checked. If the terminal is currently logged-in, the request shall be denied. If the terminal is not logged-in a message shall be displayed on the terminal requesting the user's login name. After obtaining the user's login name, the process shall disable echoing on the terminal and request the user's password. The password entered by the user shall then be encrypted and the user's login name and encrypted password shall be checked against the User Access Authentication Data Base. If there is no match, the login name and password will be requested again. Only a small number of retries shall be allowed. If this retry limit is exceeded the development contractor shall provide a mechanism for the notification of the System Security Officer. If there is a match, the processing shall continue. If the User Access Authentication Data Base indicates a login group, the group shall be checked against the Group Access Authentication Data Base and if valid, shall be used as the user's group without requiring a group password. If no group is indicated, the system default group shall be used. All users shall be initially logged in at their default security and integrity level unless that level is above the current maximum permitted on the system, in which case the user's level shall be set to the system low, and he shall be informed of his level (system low). The following functions shall be performed in logging in a user:

a. Select a terminal path, change the security and integrity levels for that path to the login level of the user, and the own of the path to the user.

b. K_spawn a process at the user's login level which will cause the user's shell to start running on the UNIX Emulator with the selected user path as the standard input and output. The use of the K_spawn primitive shall remove any special privileges possessed by the Secure Server.

3.2.1.3.2.2 Changing Group

After receiving a request to change group, a message shall be displayed on the user's terminal requesting the name of the group desired. The group name shall be checked against the Group Access Authentication Data Base to determine if the group is valid and if the user's present access level is permissible for the requested group. If a password for the group is listed the user shall be asked the password which shall be encrypted and checked. If successful, the requested group ID shall be set for the user, otherwise a rejection message shall be displayed on the user's terminal. As in login processing, only a limited number of attempts shall be permitted before notification of the System Security Officer.

3.2.1.3.2.3 Change Access Level

Upon requests by the user to change his current access level, the Server shall check that the requested level is valid for the user, group, and terminal by checking the appropriate data bases. The Server shall also verify that the requested level is permitted on the system at that time (i.e. that the requested level is at or below the current system high level). The Server shall then determine whether or not the user has an active environment (usually a shell running on a UNIX Emulator) at this level. If so, the Server shall select the user path with the previous environment. If the user does not have an active environment at this level, the Server shall attempt to find an inactive user path for that terminal and if successful, set the path level to the desired level and spawn a shell and emulator at that level.

3.2.1.3.2.4 Logout

Requests for logout shall cause a signal to be sent to all user processes invoked for that terminal requesting them to terminate. User processes which do not terminate within a reasonable time will be killed. The terminal mode (speed, parity, etc) shall be set to the values listed for it in the Terminal Profile Data Base.

3.2.1.3.2.5 Processing Operator Requests

The Access Authentication Functions shall accept and enforce commands from the System Operator to control user login, access levels, and logout.

3.2.1.3.3 Outputs

An accounting message shall be sent to the Audit Capture Process for each request even if the request was denied.

3.2.1.4 Password Modification Function

3.2.1.4.1 Inputs

The Password Modification Function shall have read/write access to the User Access Authentication Data Base and the Group Access Authentication Data Base. The Password Modification Function shall communicate with the user via the secure terminal path.

3.2.1.4.2 Processing

After receiving a request for password modification, the Secure Server shall invoke the Password Modification Function to handle the request. The Password Modifier shall turn off echoing on the user's terminal and request the user's old password which is then encrypted and checked againist the User Access Authentication Database. This check is done to insure that the person on the terminal is the real user for the logged in user ID. The user will then be requested to enter the desired new password. The user shall be requested to enter the new password a second time. If the two entries do not match the user shall be informed and the process repeated. The new password shall be encrypted and written into the User Access Authentication Data Base. The user shall be allowed to modify a group password in a similar manner if he has been designated in the Group Access Authentication Database as the Group Administrator for that group.

3.2.1.4.3 Outputs

The Password Modification Function shall write to the user terminal via the secure terminal path.

The Password Modification Function shall write new passwords in the User Access Authentication Data Base, and the Group Access Authentication Database as required.

Audit information noting that the password has been changed will be captured.

3.2.1.5 File Access Modification Function

User requests to change the access level and owner of a file shall be processed by the File Access Modification Function.

3.2.1.5.1 Inputs

The File Access Modification Function shall receive input from the user via the secure terminal path including the pathname of the file to be modified.

The File Access Modifier shall have access to status information on all files.

3.2.1.5.2 Processing

The File Access Modifier shall request from the user the path name of the file to be modified. It then checks for the existence and ownership of the file. If the requester can not access the file, or the requester is not the owner, or the requester, if not the owner, is not at ADMINISTRATOR level, or the file is open, the request shall be denied. If these checks all succeed, the user shall then be asked the operation to be performed.

For a request to change access level, checks shall be made to determine that the file exists, that the requested level is allowed on the file system on which the file resides and that the requester's access matrix includes the requested level. If these checks are successful the level of the file shall be changed.

For changing the owner of a file, the Server shall check that the user is the owner of the file or is at the ADMINISTRATOR or higher integrity level, that the file exists, that the file is not open, and that the new owner exists. If these checks are successful, the owner of the file shall be changed.

3.2.1.5.3 Outputs

The File Access Modifier shall display the current level, discretionary access information, and owner prior to changing any of this information. In addition, the proposed new values shall be displayed prior to actually changing them. In the case of a change owner a message shall be sent to the new owner informing him that he has become the file owner. All output to the user terminal shall be via the secure terminal path.

Audit information shall be gathered on each request even if the request was denied.

3.2.2 System Operation Services

3.2.2.1 Network Daemon

The computer network interface functions are spread across the UNIX Emulator, the Security Kernel and the Network Daemon. Table 1 shows the overall functional division. The remainder of this paragraph discusses the functions performed by the Network Daemon.

Table I

TCP Functional Allocation

	Security Kernel	UNIX Emulator	Network Daemon
IMP Support	x		
Rendezvous			x
Flow Control		x	
Data Stream Integrity:			
Checksum			x
Resequence		x	
Retransmission		x	
Acknowledge		x	
Duplicate Detect		x	
Routing			x
Connection Alloc.			x
Urgent Processing		x	
Formatting		x	
Security Level of Message			x

3.2.2.1.1 Users and Privileges

The Network Daemon shall be a general service program utilized by all users of the network. The Network Daemon shall be privileged to violate the security *-property to allow it to service a multi-level network. If the network is at only one security level, the Network Daemon is unprivileged.

3.2.2.1.2 Inputs

The inputs to the Network Daemon are the data streams to and from the network, and the control requests from the System Operator.

3.2.2.1.3 Processing

This paragraph discusses the functions of Table I which are performed by the Network Daemon.

3.2.2.1.3.1 Rendezvous

Rendezvous is the process of recognizing the match of connection names. For example, a remote user supplies, as part of his connection attempt, a local TCP port name to which he wishes to be connected. The process of matching the local name (on which some local process is listening) to the name being supplied by the remote process is called rendezvous. To accomplish the rendezvous, the Network Daemon shall maintain a table of connections and their names, both local and remote.

The Network Daemon shall provide the management of the internal tables necessary to provide the overall network services. These tables shall include, but are not limited to, the host status tables, the table containing the connection status for each open connection, and tables containing internal status information.

3.2.2.1.3.2 Checksum

To allow for proper internal routing, the Network Daemon shall compute the checksum of incoming messages. Messages received with an incorrect checksum shall be discarded in keeping with the TCP definition [Postel 78a] [Postel 78b]. The Network Daemon shall also compute the checksum for all outbound messages. The formula to be used for checksum calculation will be supplied by the Government.

3.2.2.1.3.3 Routing

The Network Daemon shall supply the routing information required by the network by mapping the user specified destination into the appropriate coded form.

3.2.2.1.3.4 Connection Allocation

The Network Daemon shall control the allocation of the connection resource. This allocation shall include the maintenance of internal tables describing the state of each connection. For active connections, this state also includes the information necessary to route inbound packets to the appropriate process and to fill in the fields for the destination on outbound traffic. The System Operator may manipulate the state of connections via the System Operator interface. This latter mode of operation is expected to be used infrequently.

3.2.2.1.3.5 Message Security Level

The Network Daemon shall supply the security level field of outbound messages based on the Kernel-supplied value for the level of the sending process if this field is supported by the network. If the network does not support messages of differing classification (e.g., a single level network) the Network Daemon shall not transmit messages whose security level is above the level of the network. The Network Daemon shall examine the security level of incoming messages to assure that they can be delivered to the receiving process without violation of the mandatory security policy. The security level of a message may be implicit, as in the case of single level networks.

In the case of single level networks, the Security Kernel shall provide many of the message security level functions by permitting (and denying) interprocess communication attempts to the Network Daemon. In these cases, the Network Daemon may be a process running at the level of the network. Attempts by processes of higher security level to communicate with the Network Daemon would be denied by the Security Kernel.

3.2.2.1.4 Outputs

The outputs of the Network Daemon are the data streams to and from the network and any diagnostic messages.

3.2.2.2 Line Printer Spooling

3.2.2.2.1 Users and Privileges

The Line Printer Spooler shall be an unprivileged service function available to all system users.

The Line Printer Daemon (de-spooler) shall be responsible for the printing of the appropriate classification markings on its output and shall be privileged to violate the security *-properity to remove printed spool files.

3.2.2.2.2 Inputs

The inputs to the Line Printer Spooler shall be the files to be printed (which defaults to the standard input if not supplied) and the control information discussed in section 3.1.1.2.2.2. The inputs to the Line Printer Daemon shall be the previously spooled output files and control commands from the System Operator.

3.2.2.2.3 Processing

The overall flow of processing in the Line Printer Spooler shall be as shown in Figure 3-2.

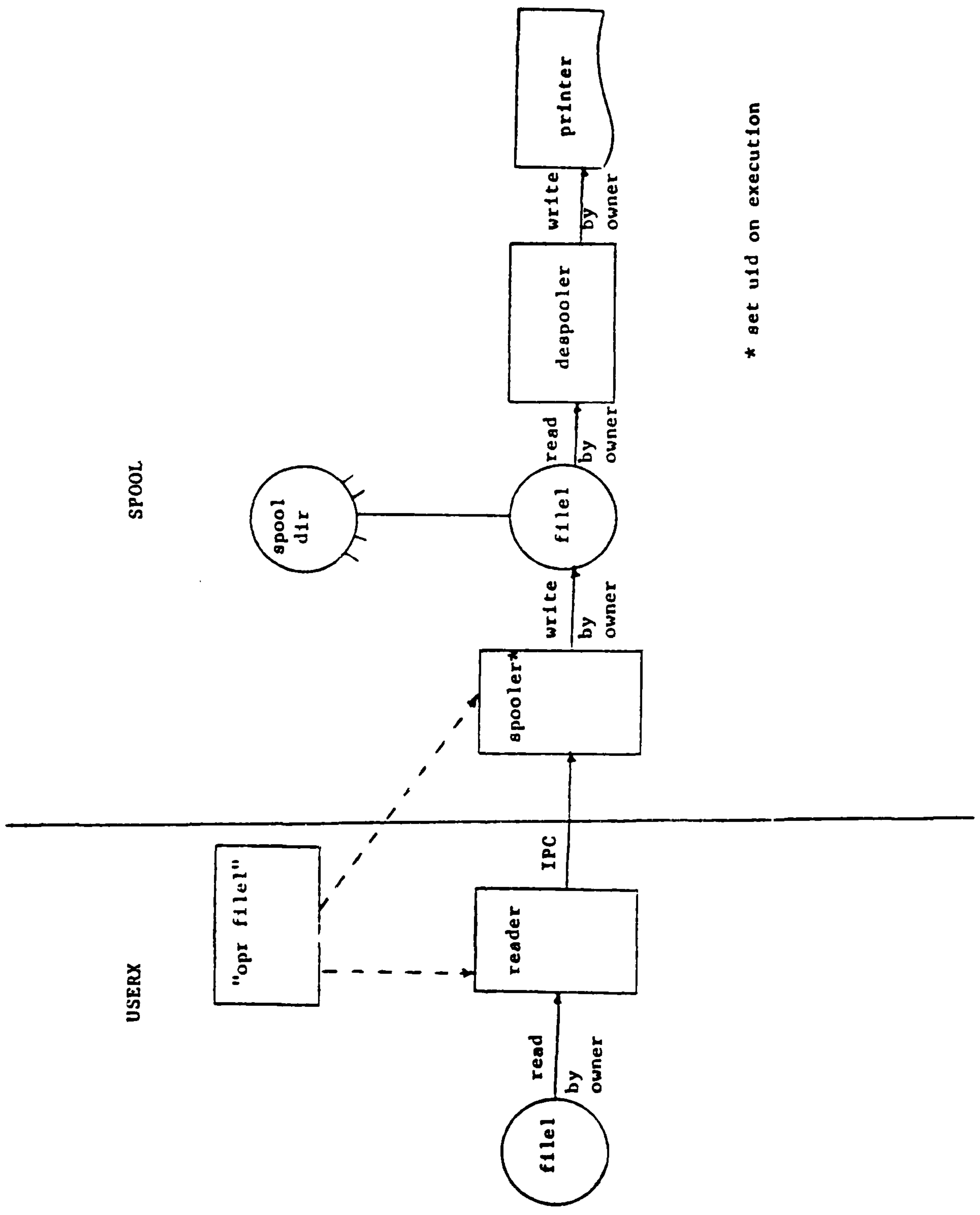

USERX
SPOOL
file1
read by owner
reader
"opr file1"
IPC
spooler*
write by owner
file1
spool dir
read by owner
despooler
write by owner
printer
* set uid on execution

Figure 1-3

The spooler shall make a copy of the files to be spooled, which, together with any control information, shall be placed in the spool directory. This directory, as well as the files it contains, shall have a discretionary access protection which disallows access to all users other than "spool", the spooler manager.

Using the corresponding control information, the Line Printer Daemon shall print each spooled file. The Daemon shall apply the appropriate headers, footers, and burst page information to correctly mark the classification of the output. Markings shall be in accordance with DoD Directive 5200.1-R. The Daemon shall create auditing records noting the creation of classified output. Finally, the Daemon shall delete the spooled output file after the printing has been successfully completed.

The Daemon shall respond to inter-process communication from the System Operator, as defined by the Daemon Control Interface above. The Daemon shall be capable of aborting the printing of a file and of backing up a page for paper changes.

3.2.2.2.4 Outputs

The output of the Spooler shall be the temporary files to be subsequently printed by the Daemon. The output of the Daemon shall be the printed output and the auditing records indicating the creation of classified output.

3.2.2.3 Assign/Deassign

3.2.2.3.1 Users and Privileges

The use of assign shall be unrestricted although the ability to assign certain devices may be restricted. The Assign and Deassign functions shall be privileged to change the owner and level of a device; i.e., to set the type independent information for the device.

3.2.2.3.2 Inputs

The user input to Assign shall specify the device(s) to be assigned. The input to Deassign shall indicate the device(s) to be deassigned. Both processes shall have access to the type independent status of devices.

3.2.2.3.3 Processing

Upon receipt of a request to assign a device, the Assign process shall check the current owner of the device. If the device is not owned by the System Resource Manager, the process will assume the device is presently assigned and disallow the request since a device can only be assigned to one user at a time. If the level of the user requesting the device is higher that the maximum level allowed for that device, the request will also be denied. To assign a device, Assign shall change the owner and access level of the device to that of the requester and the discretionary access permissions to read/write by owner.

Deassignment of a device must be requested either by the current device owner or the System Operator. The device ownership shall be return to the System

Resource Manager upon deassignment.

3.2.2.3.4 Outputs

The Assign and Deassign processes shall output messages as necessary to the user.

3.2.2.4 Mount/Unmount

3.2.2.4.1 Users and Privileges

The use of the Mount and Unmount Processes shall be unrestricted, however, the ability to mount and unmount particular file systems or to mount on particular devices may be restricted.

The Mount Process shall have the permissions needed to access the underlying devices.

3.2.2.4.2 Inputs

The user inputs to Mount shall be the name of the file system to be mounted, the drive on which the file system resides, the pathname of a presently accessible file (normally, the file would be of directory subtype), and an optional argument indicating that the file system is to be mounted read only. If no arguments are given, a list of the currently mounted file systems is given.

The process shall also read a file, maintained by the Immigration Officer Process, containing the file systems which are acceptable for mounting.

The Unmount Process inputs from the user shall be the drive to be unmounted.

3.2.2.4.3 Processing

The Mount Process shall verify that the file system is acceptable for mounting by checking that the file system is on the list maintained by the Immigration Officer Process and that it is not already mounted. Mount shall also check that the user has discretionary access to the file system (i.e., if the file system is to be mounted read/write, the user must have read/write access; if the file system is to be mounted read only, the user must have read access), that the user has write access to the parent directory, that the user's security and integrity levels are greater than or equal to the file system's level, and that the file system's set of security compartments is a subset of the user's set. If any of these checks fail, the process shall send a message to the Audit Capture Process and notify the user of the failure. If the checks are successful, the Mount Process shall issue the K_mount Kernel call which will logically mount the file system. The UNIX directory manager (UDM) ZZZ will then be called to add the file system into the UNIX structure.

The Unmount Process shall check that the user has permission to unmount the file system; i.e., the user has discretionary access to the file system, has write access to the mount on entry of the directory being unmounted, and has the appropriate security level, integrity level, and security compartments as described in the section on Mount. If these checks are successful, the Unmount Process shall issue the K_unmount Kernel call. Followed by a call to the UNIX directory manager to remove the file system from the UNIX structure.

3.2.2.4.4 Outputs

The Mount and Unmount Processes shall update a file analogous to the file /etc/mount in current UNIX which is used to inform users which file systems are presently mounted. Both processes shall provide diagnostic messages as required.

3.2.2.5 System Startup/Shutdown

3.2.2.5.1 Users and Privileges

The use of the System Startup and Shutdown mechanisms shall be restricted to the System Operator. Since the KSOS system is not in operation when the System Startup mechanism is invoked, and, thus, cannot protect itself, the System Startup mechanism shall be further restricted to be used only from the operator's console. The System Shutdown mechanism shall be similarly restricted because the KSOS system will be unavailable to protect itself after the shutdown is complete.

3.2.2.5.2 Inputs

The inputs to the System Startup Process shall be the name of the KSOS image and of the Initial Process image to be loaded. Locally specified functions may require additional inputs. The Initial Process shall then ask the user for the time, the location of the root file system, and the Secure Initiator to be used.

No inputs to the System Shutdown Process are required. Locally specified functions may require additional inputs.

3.2.2.5.3 Processing

The System Startup Process shall begin by having the ROM bootloader (or equivalent) load a KSOS bootstrap. This bootstrap shall then load the Kernel and Initial Process, and initialize the system hardware to allow proper execution of the Kernel. The Kernel will perform any internal initialization required. It will then set up internal tables, etc. to allow the Initial Process image to run as a user process, and will begin running it.

The Initial Process shall set the time of day clock, mount the root file system, and build the segments used by the Process Bootstrapper (PBB). The process shall then issue a K_invoke call to invoke the Secure Initiator.

The System Shutdown Process shall first execute a locally modifiable shell file to accomplish any desired local processing. It shall then "kill" all user processes, force all cached I/O to be completed, and call K_halt to halt the kernel.

3.2.2.5.4 Outputs

The System Startup and Shutdown Processes shall have no outputs other than diagnostic and informational messages. Sites may elect to include additional output from their locally specified startup and shutdown shell files.

3.2.2.6 Secure Mail

3.2.2.6.1 Users and Privileges

The Secure Mail mechanism shall be a service available to all users. . It shall be unprivileged.

3.2.2.6.2 Inputs

The input for the send part of Secure Mail shall be the name of a file to be mailed (which shall default to the standard input if not supplied), and the list of users to whom the mail is to be sent.

The input for the receive part of Secure Mail shall be the previously transmitted letters, if any, and an (optional) file name in which to save the mail.

3.2.2.6.3 Processing

Conceptually, Secure Mail shall be similar to "two ended spooling". The sending process shall place the mail in a "post office box" directory that is unique for each user. This directory as well as the files (letters) placed in it, shall be owned by the user "post" and shall have discretionary access protections that disallow access to all users other than "post".

The receive mail process shall retrieve the letters from the user's "post office directory" and place them in a user specified directory. So that security upgrading of letters does not occur, the user shall only be allowed to receive letters as his current security level. The mail may then be examined by the user.

3.2.2.6.4 Outputs

The output of the send part of Secure Mail shall be the letters created under the /post/to_user directories, and any diagnostic information required, such as that the specified recipient does not exist. The output of the receive part shall be a file in the user's mail directory.

3.2.3 System Maintenance Services

This class of Non-Kernel Security elated Software shall consist of those functions that provide facility for:

a. file system backup and restoration

b. file system consistency check and repair

c. system distribution, generation, and reconfiguration

3.2.3.1 File-system Backup and Restoration

The Dump and Restore functions shall provide a mechanism for the generation and recovery of incremental and complete file system dumps of KSOS files. Both Dump and Restore programs shall be trusted to allow the reading and creation of multi-level file objects.

3.2.3.1.1 Dump

3.2.3.1.1.1 Users and Privileges

Use of the Dump program shall be restricted to the Operator level and above. The program shall have the permissions needed to access the disk devices directly (rather than going through the Security Kernel and UNIX Emulator file systems).

3.2.3.1.1.2 Inputs

The inputs to the Dump program shall be similar to the inputs to the current UNIX dump program:

a. parameters which describe precisely what is to be dumped

b. the name of the file system(s) to be dumped

c. the data from the file system being dumped

3.2.3.1.1.3 Processing

After determining that the invoking user has a sufficient access level to run the Dump program, the program shall copy a KSOS file system to its output in a dump format. Dump shall be capable of producing an incremental dump of a specified KSOS file system. The incremental dump shall include all files that have changed since the last full dump, or in a specified period.

3.2.3.1.1.4 Outputs

Output of the Dump program shall be to a 9-track magnetic tape or other removable storage medium. The Dump program shall allow its output to be directed to a disk or equivalent.

3.2.3.1.2 Restore

3.2.3.1.2.1 Users and Privileges

Use of the Restore program shall be restricted to the Operator level and above. The program shall have the permissions needed to access the disk devices directly (rather than going through the UNIX Emulator file systems).

3.2.3.1.2.2 Inputs

The inputs to the Restore program shall be similar to those used by the current restor(VII) [Thompson 75] program:

a. parameters describing precisely what is to be restored

b. the name of the file system to be restored

c. the data previously dumped that is to be restored

3.2.3.1.2.3 Processing

After validating that the user is privileged to restore a file system, the Restore program shall (selectively) restore to the designated file system.

3.2.3.1.2.4 Outputs

The output of the Restore program shall consist of the restored files or file system, and any diagnostic messages produced.

3.2.3.2 File System Maintenance Services

3.2.3.2.1 Make File System

3.2.3.2.1.1 Users and Privileges

Operation of the Make File System program shall be allowed only by the System Administrator since the contents of the target pack, extent, or file system are destroyed by the initialization process. The program shall have the permissions needed to access the underlying device files, rather than going through the Security Kernel's or UNIX Emulator's file systems.

3.2.3.2.1.2 Inputs

The inputs to the Make File System program shall include the parameters which describe the initial state of the pack, extent, or file system which is being initialized.

3.2.3.2.1.3 Processing

The Make File System program shall initialize the pack, extent, or file system in accordance with the parameters supplied. The program may implement safety measures to reduce the probability of accidentally re-initializing a file system.

3.2.3.2.1.4 Outputs

The outputs of the Make File System program shall include the newly initialized pack, extent, or file system, and any diagnostic or warning messages.

3.2.3.2.2 Directory Consistency Check

3.2.3.2.2.1 Users and Privileges

The use of the Directory Consistency Check program shall be limited to the System Operator level and above since it requires access to the underlying device(s) on which the file system is built.

3.2.3.2.2.2 Inputs

The input to the Directory Consistency Check program shall include the name of the file system to be checked.

3.2.3.2.2.3 Processing

The program shall read the UNIX file system directories on the device, and compute the actual and expected link counts for each file. Counts that do not match shall be reported. In addition, any file which has an actual and an expected count of zero shall be reported.

3.2.3.2.2.4 Outputs

The outputs of the Directory Consistency Check program shall include the diagnostic messages discussed above. The program shall also report the absence no anomalies in the structure of the file system.

3.2.3.2.3 Storage Consistency Check

3.2.3.2.3.1 Users and Privileges

The use of the Storage Consistency Check program shall be limited to the System Operator level and above since it requires access to the underlying device(s) on which the file system is built.

3.2.3.2.3.2 Inputs

The inputs to the Storage Consistency Check program shall include the device to be checked and parameters to control the optional processing.

3.2.3.2.3.3 Processing

The program shall examine the named file system to determine that all allocated areas of the file system are mutually exclusive and that all storage areas are accounted for in the file system. The program shall have the capability to identify files that include named blocks.

3.2.3.2.3.4 Outputs

The output of the Storage Consistency Check program shall include a summary of the storage allocation on the device including any diagnostic information about missing or multiply allocated areas.

3.2.3.2.4 Kernel Name to Pathname Mapping

3.2.3.2.4.1 Users and Privileges

The use of the Name Mapping program shall be restricted to the System Operator because it must access the underlying device on which the file system is built.

3.2.3.2.4.2 Inputs

The inputs to the Name Mapping program shall be a list (possibly null) of Secure Entity IDentifiers (SEID's) (the Security Kernel CPCI's names for objects), and the device (file system) on which they are to be found.

3.2.3.2.4.3 Processing

If no Secure Entity Identifiers have been supplied the program shall produce a list the UNIX path names vs. SEID's for all files on the device. Otherwise, the list shall include only the requested SEID's

3.2.3.2.4.4 Outputs

The outputs of the Name Mapping program shall be the list of UNIX path names vs. SEID's, and diagnostic information as required.

3.2.3.2.5 Modify Control Entries for Filesystem

3.2.3.2.5.1 Users and Privileges

The use of the program which can modify the control entries for a file (MCE) shall be restricted to the System Operator since it requires access to the device on which the file system resides. In addition, the program should only be used by personnel with sufficient knowledge of the file system structure to be aware of exactly what may and may not be safely changed.

3.2.3.2.5.2 Inputs

The inputs to MCE shall include the file system to be modified, the node to be modified, and the fields to be changed.

3.2.3.2.5.3 Processing

The MCE program shall modify the requested fields to the requested value. The program should produce warning messages about how serious the proposed modification may be.

3.2.3.2.5.4 Outputs

The output of MCE shall be the modified file system. In addition, the file system shall be removed from the list of acceptable systems for mounting, forcing the Immigration Officer Process (3.1.4.4.4) to be invoked before the file system may be mounted. Finally, the program shall produce audit records of its use.

3.2.3.3 KSOS System Generation

Each KSOS installation can be expected to have a unique hardware configuration. Due to the requirement that KSOS must support a variety of devices, it is not practical to distribute a single "universal" copy of the system. Rather, each site will generate a system tailored to its hardware configuration.

3.2.3.3.1 Users and Privileges

The System Generation mechanism shall be designed to be used by an individual with minimal computer system expertise (e.g., the Security Officer who must certify the resulting system). The function shall possess no special privileges. However, it must be recognized that access to the KSOS system image as it is being built is tantamount to being able to violate all the rules of the system.

3.2.3.3.2 Inputs

The KSOS system shall be distributed as a three-component package. The first component, a distribution tape, shall contain a starter system and standard system-build libraries to be copied to the target system disk at the remote site. Documentation for KSOS shall comprise the second component of the distribution. The third element of the distribution package shall consist of the system generation instructions and generation password sequence. The distribution tape and generation package shall be transmitted separately as SECRET-level material. The distribution tape will normally be sent to the site manager, while the installation guide and passwords will be sent to the System Security Officer.

A checksum technique shall be used to validate transfer of the Kernel components during the system build cycle. Each copy of the Kernel modules shall have a unique identifier (probably derived from the target machine's serial number) that will generate unique module checksums for each distribution tape. Each element in the distributed system shall contain a component checksum. An interactive system-build program shall compute the checksum associated with each component, notifying the system builder (Security Officer) of the checksum failure and identifying the failed component. As a general rule, systems built from "failed" modules cannot be certified as multi-level secure. Using these concepts, the required integrity can be provided to remotely generated systems without requiring the System Security Officer to be a system programmer.

The standard KSOS distribution tape shall contain a bootstrap loader, a site-specific starter system used to generate the target KSOS system and all modules required to generate a KSOS system, at a remote site.

3.2.3.3.3 Processing

3.2.3.3.3.1 Bootstrap

The bootstrap loader shall be the first entry on the KSOS distribution tape. A small program entered from the machine console (or executed from a standard ROM loader) shall read the bootstrap which shall, in turn, read the starter system from the tape, and then transfer control to the starter system. Each of the tape-based programs shall be checksummed to detect media failure.

3.2.3.3.3.2 Starter System

The starter system shall be a KSOS system previously generated for a particular minimum configuration. The starter system shall require no more than the minimum hardware specified in the System Specification (Type A). The starter system may be specific to a particular hardware configuration. When the distribution tape is generated by the maintenance and support organization, the appropriate starter system shall be selected. Alternatively, the distribution tape may contain starter systems for several common configurations. The device and vector addresses of the disk drive and console shall have standard values. Any other configuration shall be handled as a special case by the maintenance and support organization.

3.2.3.3.3.3 Libraries

The starter system shall transfer the KSOS system programs, libraries, and archives from the distribution tape to the specified system disk to complete the system restoration process. Media failures should be detected through component checksums.

3.2.3.3.3.4 Special Cases

Unique configurations with no magnetic tape or non-standard system disks or console addresses shall be handled as special cases. Installations without tape drives shall receive a disk-based distribution of KSOS.

3.2.3.3.3.5 System Generation

System generation is basically a matter of selectively including the appropriate device drivers in the Security Kernel, defining the /dev files and their underlying Kernel representations and "binding" the /dev files to the Kernel's assignment structure. The remainder of the system software may require minor local (per site) modifications, for example to set the local time zone. All such common local modifications shall be automated through the use of shell scripts or other equivalent mechanisms.

The Security Officer will execute an interactive System Configurator program similar to mkconf(VIII) [Thompson 75] for standard UNIX. The System Configurator program may be executed under the starter system or under a larger KSOS configuration. The Security Officer shall interact with the program to define the hardware configuration for which the KSOS system is to be generated. A list of hardware parameters (e.g., in the PDP-11/70 version, base address of the control and status registers, vector addresses, and interrupt priorities) must be provided to the Security Officer by a knowledgeable individual. A "standard" list shall be provided in the installation instructions.

The System Configuration dialogue shall allow:

a. selection of devices.

b. specification of device control registers, and vector addresses, and interrupt priority levels.

c. device driver source/entry names.

The System Configurator program shall prompt with "standard" device information, unless a local configuration file is specified. In that case, the Configurator program shall be used to create an updated (new) configuration file. The new configuration file shall be checked for consistency, and then used to generate the "new" local system by comparison of the new and old configurations. Any changes shall cause the appropriate drivers to be edited, recompiled, checksummed, and archived. Appropriate controls on use of the Configurator program must be observed, since portions of the Security Kernel will have to be recompiled to change device register addresses.

3.2.3.3.3.6 Integrity of the Inputs

The configuration procedure shall execute the internal program that validates the checksum of each of the modules required in the link edit phase. In the event of a checksum error, the failed modules shall be identified to the system builder (Security Officer). If any checksum failures are noted, the system shall not be certified as being suitable for multi-level secure operation.

3.2.3.3.3.7 Rename System

The final operation of the configuration procedure shall give the system its bootstrap name, e.g. KSOS-1.0. This step should not occur if any errors (e.g., checksum errors) have occurred. This procedure is executed prior to the Security Officer bootstrapping up the new version of the KSOS system. For additional security and integrity it is recommended that sites do not leave the sources for the trusted software on normally on-line disks.

3.2.3.3.3.8 System Regeneration

When new hardware is added to a configuration, the system will need to be "regenerated" to reflect the changes. This regeneration process should be carried out by the Security Officer in a single-user environment. By using the Configurator program in an update mode, only new information or changes will need to be entered to create the new system configuration. While the procedure could easily be carried out in a multi-user environment, single-user operation appears to be an appropriate mode of operation during system regeneration of KSOS.

3.2.3.3.4 Outputs

The output of the system (re)generation process shall be a "bootstrappable" KSOS image, and any diagnostic or status information required.

3.2.4 System Administration Services

3.2.4.1 User Control Editor

The User Control Editor shall be used by the System Administrator to control access to the system by modifying the User Access Authentication Data Base and the Group Access Authentication Data Base.

3.2.4.1.1 Users and Privileges

Use of this program shall be restricted to users running at the system high security and integrity level and in the system compartment.

This process shall be privileged to issue K_secure_terminal_lock calls.

3.2.4.1.2 Inputs

The User Control Editor shall receive input from the Administrator via the secure path. It shall have read/write access to the User Access Authentication Data Base and to the Group Access Authentication Data Base. The User Control Editor shall also have read access to the Security Map Data Base.

3.2.4.1.3 Processing

The User Control Editor shall process requests from the System Administrator to add and delete users, to change a user's default login level, to change a user's access level capabilities, to change a user's initial login group, to change user and group passwords, and to assign Group Administrators. Appropriate checks shall be made by the process to assure data base integrity. Audit messages shall be generated and sent to the Audit Capture Process.

3.2.4.1.4 Outputs

The User Control Editor shall send such messages as necessary to the user terminal via the secure terminal path. It shall also make changes as necessary to the User Access Authentication Data Base and the Group Access Authentication Data Base. In addition, it shall generate and send to the Audit Capture Process necessary audit trail messages.

3.2.4.2 Audit Capture Process

The Audit Capture Process shall be a system process which receives audit messages and writes them onto a accounting file.

3.2.4.2.1 Users and Privileges

The Audit Capture Process shall be a system process having no direct users.

3.2.4.2.2 Inputs

The Audit Capture Process shall receive accounting information from the Kernel and NKSR processes via the inter-process communication mechanism and the signal mechanism.,

The Audit Capture Process shall accept messages sent through the signal mechanism from processes running at OPERATOR integrity level or higher to execute certain System Operator commands.

3.2.4.2.3 Processing

All messages received by the Audit Capture Process shall be marked as to whether they were sent using the signal mechanism or the inter-process communications mechanism. The message type shall then be checked to determine if the type is valid. The information may be reformatted and additional information such as a time stamp added before the message is written on the accounting file.

The Audit Capture Process shall accept requests from the System Operator to switch accounting files.

3.2.4.2.4 Outputs

After processing, accounting records shall be written on the audit file.

3.2.4.3 Privilege Control Process

The Privilege Control Process shall provide the Security Officer with the ability to to grant and revoke special privileges to processes.

3.2.4.3.1 Users and Privileges

The Privilege Control Process shall be a restricted use process run by the System Administrator at system high security and integrity level and in the system compartment.

The Privilege Control Process shall have the ability to violate the *-property for security and integrity, and discretionary access.

3.2.4.3.2 Inputs

The Privilege Control Process shall receive input from the Security Officer via the terminal. It shall also be able to obtain status information on files.

3.2.4.3.3 Processing

The Privilege Control Process shall receive requests from the System Security Officer to grant and revoke special privileges to processes by changing elements of the object type dependent information for the file from which the process is invoked.

3.2.4.3.4 Outputs

The process shall write to the Security Officer via the secure terminal path. It shall change file status information as necessary. The Privilege Control Process shall also send audit messages as necessary.

3.2.4.4 Immigration Officer Process

The Immigration Officer Process shall maintain a data base of file systems may which be mounted on the system.

3.2.4.4.1 Users and Privileges

The Immigration Officer Process shall be a restricted use process and can be run only by a user at the ADMINISTRATOR integrity level.

The Immigration Officer Process shall have the permissions needed to access the underlying disk devices.

3.2.4.4.2 Inputs

The inputs to the Immigration Officer Process shall consist of the current version of the mountable file systems data base, parameters which specify which file system is to be immigrated, and the data making up the file system.

3.2.4.4.3 Processing

The Immigration Officer Process shall check that no privileged software resides on the volume. The development contractor may include additional checks. If all checks are successful, the program shall perform any remapping of security level representations that has been requested. The Immigration Officer shall first check that the proposed remapping is valid: e.g., that this system includes all of the access levels represented on the foreign file system. If all the checks and the remapping (if required) are successful, the immigration officer data base shall be updated to reflect the fact that the volume is acceptable for mounting.

3.2.4.4.4 Outputs

The outputs of the Immigration Officer Process shall be the updated immigration officer data base and any diagnostic messages required.

3.2.5 Special Requirements

3.2.5.1 Human Performance

This paragraph is not applicable to this specification.

3.2.5.2 Government Furnished Property List

The compiler used for the Non-Kernel Security Related Software may be furnished by the Government.

3.3 Adaptation

The Non-Kernel Security Related Software should require no per-site modification. Individual sites may, however, modify the non-trusted functions as they see fit. Field maintenance of the trusted functions shall be prohibited unless subject to the quality assurance procedures (e.g.formal specification, verification, etc) equivalent to its initial creation.

4. QUALITY ASSURANCE PROVISIONS

4.1 Introduction

The underlying philosophy of KSOS quality assurance is to provide a convincing demonstration of the security and completeness of the system. Testing and quality assurance are a complement to the formal verification aspects of KSOS. All testing on KSOS shall be conducted at the development contractor's facility.

KSOS shall be subject to the technical reviews and audit procedures of MIL-STD-1521A, Appendices A-F. The following technical reviews and audits will be conducted:

a. System Requirements Review (Appendix A)

b. System Design Review (Appendix B)

c. Preliminary Design Review (Appendix C)

d. Critical Design Review (Appendix D)

e. Functional Configuration Audit (Appendix E)

f. Physical Configuration Audit (Appendix F)

The Critical Design Review shall include a review of any mathematical proofs showing that the design meets the requirements of the Government approved DoD security model.

The Functional Configuration Audit shall be the vehicle for the formal review of the design versus the mathematical description. The Physical Configuration Audit shall be the vehicle for the formal review of the "as built" computer programs versus both their design and their supporting documentation. In addition, the Physical Configuration Audit shall verify that the implementation requirements of the System Specifications (Type A) have been met. Both audits shall include review of any relevant mathematical proofs of correctness.

4.1.1 Category I Testing

The development contractor shall establish test plans to demonstrate that all the requirements of Section 3 have been met. Because this CPCI is composed of a number of autonomous functions, the primary vehicle for this demonstration shall be the Preliminary Qualification Tests. These tests shall be designed to exercise all the interface options for each function. To the extent feasible, the tests shall exercise all combinations of options. The tests shall include sequences intended to fail, such as attempts to violate the system's security rules or to use unacceptable combinations of options. Government approval of all test plans is required.

The test sequences for trusted functions shall be designed with the same rigor as those for the Security Kernel. Specifically, the test sequence should attempt to assure that all statements of the program have been exercised.

The general mechanism for assuring that the requirements of Section 3 have been met shall be a comparison of the system "state" before and after the function has been invoked. To ensure repeatability, all testing shall be automated if possible. Such automation may include (but is not limited to) shell scripts, test programs, and terminal emulation by another computer system.

4.1.2 Category II Testing

The KSOS system shall be subject to an overall system test (Category II) in accordance with the System Specification (Type A) and the development contract. This Category II test shall demonstrate that the three CPCI's which make up the KSOS system function correctly together. The Category II testing shall attempt to represent a typical user load including at least ten (10) on-line terminals and a network connection. To the extent feasible, the Category II testing shall also be automated through the use of shell scripts, and (optionally) another computer system emulating the terminal load.

Table II - Quality Assurance

Assurance techniques:

NA	Not Applicable
V	Verification
F	Formal Specification
T	Testing
D	Demonstration
A	Analysis
I	Inspection

Applicable test types:

1	Module level testing
2	Integration testing
3	Acceptance testing

Requirement	Assurance Techniques							Test types		
	NA	V	F	T	D	A	I	1	2	3
3.1.1.1.				x	x	x	x	x	x	x
3.1.1.2.1.1.			x	x	x	x	x	x	x	x
3.1.1.2.1.2.			x	x	x	x	x	x	x	x
3.1.1.2.2.1.1.			x	x	x	x	x	x	x	x
3.1.1.2.2.1.2.				x	x	x	x	x	x	x
3.1.1.2.2.1.3.				x	x	x	x	x	x	x
3.1.1.2.2.2.			x	x	x	x	x	x	x	x
3.1.1.2.2.3.			x	x	x	x	x	x	x	x
3.1.1.2.2.4.			x	x	x	x	x	x	x	x
3.1.1.2.2.5.				x	x	x	x	x	x	x
3.1.1.2.2.6.				x	x	x	x	x	x	x
3.1.1.2.4.1.				x	x	x	x	x	x	x
3.1.1.2.4.2.				x	x	x	x	x	x	x
3.1.1.2.4.3.				x	x	x	x	x	x	x
3.1.1.2.4.4.				x	x	x	x	x	x	x
3.2.1.1.			x	x	x	x	x	x	x	x
3.2.1.2.			x	x	x	x	x	x	x	x
3.2.2.1.			x	x	x	x	x	x	x	x
3.2.2.2.			x	x	x	x	x	x	x	x
3.2.2.3.			x	x	x	x	x	x	x	x
3.2.2.4.			x	x	x	x	x	x	x	x
3.2.2.5.				x	x	x	x	x	x	x
3.2.2.6.				x	x	x	x	x	x	x
3.2.3.1.1.				x	x	x	x	x	x	x
3.2.3.1.2.				x	x	x	x	x	x	x
3.2.3.2.1.				x	x	x	x	x	x	x
3.2.3.2.2.				x	x	x	x	x	x	x
3.2.3.2.3.				x	x	x	x	x	x	x
3.2.3.2.4.				x	x	x	x	x	x	x
3.2.3.2.5.				x	x	x	x	x	x	x
3.2.3.3.				x	x	x	x	x	x	x

3.2.4.1.	x	x	x	x		x	x	x
3.2.4.2.	x	x	x	x		x	x	x
3.2.4.3.	x	x	x	x		x	x	x
3.2.4.4.	x	x	x	x		x	x	x

5. PREPARATION FOR DELIVERY

This section is not applicable to this specification.

6. NOTES

This section is not applicable to this specification.

Appendix A: MAJOR CHANGES TO THE DESIGN

The main body of this document outlines the intended design of the NKSR portion of KSOS. During the implementation process, however, some aspects of the design have changed. This appendix gives a brief outline of the major changes in the design. Comments are given below by CPC and are in order of the CPC number.

The basic combined functionality for the Secure Initiator (CPC 3.1.1) and the Secure Server (CPC 3.1.2) has not changed between the previous version of the B5's and the current version. This functionality, however, was reproportioned between the two processes and the B5 sections were written to reflect this change.

At this time, Login (CPC 3.1.3) does not have the mechanism for locking out terminals.

The Password Modification Function (PMF), like Login and Logout, shall be contained within the code of the Secure Server. This function, however, is not currently implemented.

Level Preserving Copy (CPC 3.1.8) and Level Preserving Print (CPC 3.1.9) are implemented but not yet debugged or tested.

The status of the Network Daemon (CPC 3.2.1) can be found in the final report for KSOS.

Some changes were made to the checks Mount (CPC 3.2.5) performs. Mount now checks that the mount on entry has not been mounted on already. The other change is that when Mount checks (using SMXflow) that the user has flow, in the case of security, to the file system to be mounted. In the case of integrity, the flow check is made as if the user had the privilege to violate the integrity *-property.

System Shutdown (CPC 3.2.8) has not been implemented. Currently shutdown consists of pressing the halt switch.

Secure Mail (CPC 3.2.9) needs to be redesigned tc take into account changes made to the Kernel design (in particular, those changes dealing with the set uid problem).

The B5's for the UNIX Directory Manager (CPC 3.2.11) are located in the emulator B5's for historical reasons.

Make File System (or File System Initialization) (CPC 3.3.3) is implemented as three programs: the Pack Initialization program (PKI), the Extent Initialization program (EXI), and the Boot Copy program (BTCP).

The Directory Consistency Check (CPC 3.3.4) and the Kernel Name to Pathname Mapping (CPC 3.3.6) have not been implemented.

Modify File System Control Entry (CPC 3.3.7) does not print warning messages indicating the seriousness of the proposed changes. It also does not remove the file system from the immigration officer data base. If this function were to be implemented, it could cause problems when trying to make modifications to the root file system. The problem lies in the fact that the root file system must be mounted to run the immigration officer and to access the immigration officer data base.

The User Control Data Base Editor (CPC 3.4.1) is implemented as two programs: a program to change user information, and a program to change group information. Also, there is no K_secure_terminal_lock_call as is mentioned.

At this time, all messages sent to the Audit Capture Process (3.4.2) are kept. They are not marked as to whether they were sent via the signal mechanism or the inter-process communication (ipc) mechanism. In addition, the messages are not reformatted, though a time stamp is appended.

A program called the Audit Capture Operator Interface (ACP_OP) is used to send messages from the Operator to the Audit Capture Process. These messages cause the Audit Capture Process to report the name of the current audit file, to switch to a new audit file, or to place the audit messages out on the console instead of a file (or vice versa). In addition, the interface will remove or print any audit file indicated. When printing the audit file, the interface will format the messages in a readable form.

The Privilege Control Process (CPC 3.4.3) has not been implemented. Its functionality, however, is performed by the Modify File System Control Entry (CPC 3.3.7).

The list of possible privileges which could be given or withdrawn by the process has changed. Below is a brief description of each privilege currently permitted:

a. The ability to violate the simple security property. (This privilege is not intended to be used. Originally, it was include merely for completeness. It was found, however, to be useful during implementation to circumvent certain types of bugs and allow development to continue without having to wait for the bug to be fixed.)

b. The ability to violate the simple integrity property.

c. The ability to violate the *-property for security.

d. The ability to violate the *-property for integrity.

e. The ability to violate the security compartments.

f. The ability to violate discretionary access.

g. The ability to use the K_link and K_unlink primitives.

h. The ability to use the K_mount and K_unmount primitives.

i. The ability to change the following pieces of type independent information about an object.

1. The security level of the object.
2. The integrity level of the object.
3. The security compartments of the object.
4. The owner of the object.
5. The group of the object.

j. The ability to save the process image in the swapping area.

k. The ability to lock a segment in core.

l. The ability to use the K_signal primitive.

m. The ability to use the K_halt primitive.

n. The ability to do volume valids through the K_device_function primitive.

o. The ability to walk through the process table.

p. The ability to use the execute bit to determine if one has read access to the object (used, for example, when searching directory structures).

q. The ability to change the terminal path.

r. The ability to change characteristics of a terminal; e.g., the speed or the parity of a terminal.

s. The ability to grant or revoke privileges using the K_set_priv primitive.

The Immigration Officer Process (CPC 3.4.4) is not implemented in full. It was planned to take the File System Restore program (CPC 3.2.2) and add the necessary checks of the file system and the handling of the immigration officer data base to it.

```
$("     MODULE          System Operation Services Specs
                                 (Version 1.8, 11:48:07)
                        Module Name (NSO.specs)
                        File Name (/kr/sunix/specs/nksr/s.NSO.specs)
        CONTENTS        Assign, Deassign, Mount, Unmount
        TYPE            SPECIAL.specifications
        LAST CHANGED    4/21/81,11:48:07
")

MODULE SOS
    TYPES

nonDisType: STRUCT_OF(
                INTEGER securityLevel; SET_OF securityCat securityCatS;
                INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
  $(integrityCat is typically the null set)
daType: SET_OF daMode;
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
tiiStruct: STRUCT_OF(nonDisType nd;
                modeStruct da; INTEGER owner, group; SET_OF privType priv);
DBMEentry:  STRUCT_OF(VECTOR_OF CHAR fileSysName;
        VECTOR_OF CHAR mountOnName; seid mountOnSeid; fileNsp slot;
        BOOLEAN readOnly);
        $("In addition to the above fields, the implementation also includes
        a timestamp, omitted from these specs for simplicity,
        and TIIinfo(fileSysName), which is omitted here since
        there is no need to include TIIinfo() as a specific field
        in a STRUCT within the specs.")

DDBentry:  STRUCT_OF(nonDisType maxLevel;
                    BOOLEAN unloaded,
                            inUse);

deviceProfileEntry:  STRUCT_OF(seid devSeid; nonDisType maxLevel);

    $(from fca)
fileNsp:  {129 .. 255};
globalData: STRUCT_OF(INTEGER linkCount, openCount, timeLastMod; seid subtype;
                      BOOLEAN openAtCrash);
  $(state information for all openable objects)

mountTableEntry: STRUCT_OF(seid rootSeid; BOOLEAN readOnly;
                           tiiStruct devTii; globalData devGl);

    PARAMETERS

INTEGER  unassignedUserId $(For use by deassign);
INTEGER  operatorIntegrityLevel;
INTEGER  maxSizeDBME;
seid nullSeid;
fileNsp rootFileNsp;  $(nullSeid and rootFileNsp are for initializing DBME)
SET_OF deviceProfileEntry deviceProfile;
        $(deviceProfile is determined at system generation time;
        for each "allowable" device devSeid, it provides the maximum
        level of the device.  Used in initializing DDB below.)
SET_OF VECTOR_OF CHAR DBIOset;
```

```
        $(DBIOset is determined at system generation time;  contains
        the name of all extents that are allowed to be mounted.  Used
        here in initializing DBIO.)

    DEFINITIONS

BOOLEAN LE(nonDisType ans,x) IS
        ans.securityLevel <= x.securityLevel            AND
        ans.securityCatS SUBSET x.securityCatS          AND
        ans.integrityLevel <= x.integrityLevel          AND
        ans.integrityCatS SUBSET x.integrityCatS;
  $(The basic ordering relation between two nonDisTypes)

    EXTERNALREFS

    FROM smx:
INTEGER SENlowLevel;
INTEGER SENhighLevel;
seid: DESIGNATOR;
secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
                   tExtent, tNull};
privType: {
        privFileUpdateStatus,   privLink,       privLockSeg,
        privModifyPriv,         privMount,
        privSetFileLevel,       privSetSegProcLevel,
        privStickySeg,          privTerminalLock,
        privViolSimpSecurity,   privViolStarSecurity,
        privViolSimpIntegrity,  privViolStarIntegrity,
        privViolDiscrAccess,    privSignal,     privWalkPTable,
        privHalt,               privKernelCall, privViolCompartments,
        privRealizeExecPermissions};

daMode: {daRead, daWrite, daExecute};
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;
domainType: {userDomain, supervisorDomain};

VFUN TIIinfo(seid s) -> tiiStruct st;
VFUN SMXflow(seid pSeid, oSeid; daType da) -> BOOLEAN b;          $(SMXflow)
VFUN SMXdap(seid pSeid, oSeid; daType da) -> BOOLEAN b;
VFUN SENseidNsp(seid anySeid) -> INTEGER nsp;
VFUN SENseidToInt(seid anySeid) -> INTEGER i;
VFUN SENseidType(seid s) -> secureEntityType set;             $(SENseidType)
VFUN SMXhasPriv(seid pSeid; privType privl) -> BOOLEAN b;       $(SMXhasPriv)

     FROM fca:
VFUN FCAinfo(seid fSeid) -> globalData gl;                        $(FCAinfo)
VFUN FCAmountTable(fileNsp nsp) -> mountTableEntry mte;  $(FCAmountTable)

     FROM udm:
OFUN mount(seid pdirSeid,nspSeid; VECTOR_OF CHAR name);
OFUN unmount(seid pdirSeid; VECTOR_OF CHAR name);

     FROM ker:
OVFUN K_mount(seid dev; BOOLEAN readOnly) -> fileNsp newNsp;  $(K_mount)
OFUN K_unmount(seid devSeid);
```

```



    FUNCTIONS

$(------------------   VFUNS   -----------------------)
VFUN DDB(seid dSeid) -> DDBentry ans;
        $(Device Data Base)
    HIDDEN;
    INITIALLY
       ans =
        (LET deviceProfileEntry x |( x INSET deviceProfile
                AND x.devSeid = dSeid)
         IN
         IF x ~= ?
         THEN
           STRUCT(x.maxLevel, FALSE, FALSE)
         ELSE
           ?);

VFUN DBIO() -> SET_OF VECTOR_OF CHAR ans;
        $(Data Base Immigration Officer - The name of the extent you
        are trying to mount in nksrMount must be in DBIO)
    HIDDEN;
    INITIALLY
    ans = DBIOset;

VFUN DBME() -> SET_OF DBMEentry ans;
  $(Data base of mounted extents.
  Initialized in the implementation by Secure Initiator;  the specs
  below simulate this initialization.  Entries are added
  by nksrMount, deleted by nksrUnmount)

    HIDDEN;
    INITIALLY
      ans = {STRUCT(
        "/",                    $(root fileSysName)
        "",                     $(null mountOnName)
        nullSeid,               $(null mountOnSeid)
        rootFileNsp,
        FALSE
        )};


$(------------------   OFUNS   -----------------------)


OFUN assign(seid devSeid)[seid pSeid];

      DEFINITIONS
  tiiStruct ptii IS TIIinfo(pSeid);

      EXCEPTIONS
  notDevice:  SENseidType(devSeid) ~= tDevice;
  badDev:  DDB(devSeid) = ?;
  badLevel:  NOT LE(ptii.nd,DDB(devSeid).maxLevel);
```

```
 inUse:  DDB(devSeid).inUse;

   EFFECTS
 'TIIinfo(devSeid) = STRUCT(
    ptii.nd,
    STRUCT({daRead,daWrite},{},{}), $(read/write by owner only)
    SENseidToInt(pSeid), $(new owner is pSeid)
    ptii.group,
    ptii.priv);
 'DDB(devSeid) = STRUCT(
    DDB(devSeid).maxLevel,
    TRUE, $(device is now "unloaded")
    TRUE);  $(device is now in use)

OFUN deassign(seid devSeid) [seid pSeid];

   DEFINITIONS
 tiiStruct dtii IS TIIinfo(devSeid);
 tiiStruct ptii IS TIIinfo(pSeid);

   EXCEPTIONS
 notDevice:  SENseidType(devSeid) ~= tDevice;
 badDev:  DDB(devSeid) = ?;
 badOwner:  dtii.owner ~= SENseidToInt(pSeid) AND
       ptii.nd.integrityLevel < operatorIntegrityLevel;
       $(Either the invoking process owns the device, or is
       at or above operator integrity level)

   EFFECTS
 'TIIinfo(devSeid) = STRUCT(
    STRUCT(SENhighLevel,{},SENhighLevel,{}),
    STRUCT({},{},{}),
    unassignedUserId,
    unassignedUserId,
    {}
    );

 'DDB(devSeid) = STRUCT(
    DDB(devSeid).maxLevel,
    DDB(devSeid).unloaded,
    FALSE
    );

OFUN nksrMount( VECTOR_OF CHAR mountFromName;
        seid mountFromSeid;
        VECTOR_OF CHAR mountOnName;
        seid mountOnSeid;
        BOOLEAN readOnly
        ) [seid pSeid];

 $(The parameters mountFromSeid and mountOnSeid are given explicitly
 in these specs for convenience.  In the implementation, these seids
 are determined from mountFromName, drive number, and mountOnName resp.)

 $("Between the kernel and the nksr, there are three mount functions.
 K-mount performs physical mounting;  mount in the directory manager
performs a simple mapping in the directory VFUN: the present
nksrMount utilizes the functionality of the other two mounts.
As a result of nksrMount, an extent is now interpreted as a file system
by the directory manager.")

    DEFINITIONS
fileNsp openSlot IS
  SOME fileNsp freeMountNsp | FCAmountTable(freeMountNsp)=?;
$(openSlot simulates the value returned by K_mount;
the related value nspSeid, defined next, is used in the
"call" to mount in the EFFECTS section below.)

seid nspSeid IS
  SOME seid s | SENseidNsp(s) = openSlot;
  $(Need a seid, rather than simply an nsp component to pass to mount)

   EXCEPTIONS
alreadyMounted:  EXISTS DBMEentry x INSET DBME() :
      x.fileSysName = mountFromName;
badExtent:  NOT mountFromName INSET DBIO():  $(The extent you are trying
              to mount must be in the Immigration Officer database, DBIO)
DBMEfull:  CARDINALITY(DBME()) = maxSizeDBME;
badRWAccess:  NOT readOnly AND NOT SMXflow(pSeid, mountFromSeid,
                      {daRead,daWrite});
badRAccess:  readOnly AND NOT SMXflow(pSeid,mountFromSeid,{daRead});
badWAccess:  NOT SMXflow(pSeid, mountOnSeid, {daWrite});
badRWdap:  NOT readOnly AND NOT SMXdap(pSeid, mountFromSeid,
                      {daRead,daWrite});
badRdap:  readOnly AND NOT SMXdap(pSeid,mountFromSeid,{daRead});
badWdap:  NOT SMXdap(pSeid, mountOnSeid, {daWrite});
notStarSec:  NOT SMXhasPriv(pSeid,privViolStarSecurity);
$(Exceptions from FCAmount follow)
XbadPriv:  NOT SMXhasPriv(pSeid,privMount);
XnoFile:  FCAinfo(mountFromSeid) = ?;  $(Successfully passing the
              badExtent check above implies that this check will
              be passed successfully)
XnoClass:  SENseidType(mountFromSeid) ~= tExtent;$(same comment as XnoFile)
XmntSpace:  RESOURCE_ERROR;

EXCEPTIONS_OF mount(mountOnSeid,nspSeid,mountOnName);

    EFFECTS
openSlot = EFFECTS_OF K_mount(mountFromSeid,readOnly);
EFFECTS_OF mount(mountOnSeid,nspSeid,mountOnName);
'DBME() = DBME() UNION {STRUCT(
      mountFromName,
      mountOnName,
      mountOnSeid,
      openSlot,
      readOnly
      )};


OFUN nksrUnmount(seid fSeid) [seid pSeid];
  $(fSeid corresponds to mountOnSeid from nksrMount)

```

```
    DEFINITIONS
DBMEentry x IS SOME DBMEentry y | (y INSET DBME()  AND
  y.mountOnSeid = fSeid);
  $(It is assumed that mountOnSeid, as a field within DBMEentry,
  is a key, i.e., uniquely determines the entire entry y.)
BOOLEAN readOnly IS x.readOnly;
fileNsp fsNsp IS x.slot;

    EXCEPTIONS

NotalreadyMounted:  x = ?;
badWAccess:  NOT SMXflow(pSeid, fSeid, {daWrite});
badWdap:  NOT SMXdap(pSeid, fSeid, {daWrite});
notStarSec:  NOT SMXhasPriv(pSeid,privViolStarSecurity);
XbadPriv: NOT SMXhasPriv(pSeid, privMount);
XnoXXX: FCAmountTable(fsNsp) = ?;
XnoTranquil:
  EXISTS seid fSeid | SENseidNsp(fSeid) = fsNsp
    : FCAinfo(fSeid).openCount > 0;
EXCEPTIONS_OF unmount(fSeid, x.mountOnName);

    EFFECTS

EFFECTS_OF K_unmount(fSeid);
EFFECTS_OF unmount(fSeid, x.mountOnName);
'DBME() = DBME() DIFF {x};


END_MODULE
```

```
$("     MODULE           Secure User Services Specs
                                 Version (1.7,14:52:13)
                         Module Name (NSU.specs)
                         File Name (/kr/sunix/specs/nksr/s.NSU.specs)
        CONTENTS         Secure Server, Login, Logout, ChangeAccess Level,
                         ChangeGroup, FileAccessModifier
        TYPE             SPECIAL.specifications
        LAST CHANGED     11/21/80,14:52:13
")


MODULE  SUS

    TYPES

nonDisType: STRUCT_OF(
               INTEGER securityLevel; SET_OF securityCat securityCatS;
               INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
  $(integrityCat is typically the null set)
daType: SET_OF daMode;
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
tiiStruct: STRUCT_OF(nonDisType nd;
               modeStruct da; INTEGER owner, group; SET_OF privType priv);


    PARAMETERS

INTEGER  sysHighLevel  $(Highest possible security and integrity level);
SET_OF securityCat allSecCats  $(Full universe of securityCatS)   ;
SET_OF integrityCat allIntCats  $(Full universe of integrityCatS) ;



    DEFINITIONS

nonDisType minimum(SET_OF nonDisType sd) IS
    SOME nonDisType ans | ans INSET sd                             AND
      (FORALL nonDisType x INSET sd:
        ans.securityLevel <= x.securityLevel                      AND
        (FORALL securityCat a INSET ans.securityCatS : a INSET
                                      x.securityCatS)              AND
        ans.integrityLevel <= x.integrityLevel);

BOOLEAN LE(nonDisType ans,x) IS
        ans.securityLevel <= x.securityLevel              AND
        ans.securityCatS SUBSET x.securityCatS            AND
        ans.integrityLevel <= x.integrityLevel            AND
        ans.integrityCatS SUBSET x.integrityCatS;
  $(The basic ordering relation between two nonDisTypes)



    EXTERNALREFS

    FROM smx:
seid: DESIGNATOR;
```

```
secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
                   tExtent, tNull};
privType: {
        privFileUpdateStatus,   privLink,       privLock,
        privModifyPriv,         privMount,
        privSetFileLevel,       privSetSegProcLevel,
        privStickySeg,          privTerminalLock,
        privViolSimpSecurity,   privViolStarSecurity,
        privViolSimpIntegrity,  privViolStarIntegrity,
        privViolDiscrAccess,    privSignal,     privWalkPTable,
        privHalt,               privKernelCall, privViolCompartments,
        privRealizeExecPermissions};

daMode: {daRead, daWrite, daExecute};
securityCat: DESIGNATOR:
integrityCat: DESIGNATOR;
domainType: {userDomain, supervisorDomain};

VFUN TIIinfo(seid s) -> tiiStruct st;
VFUN SMXflow(seid pSeid, oSeid; daType da) -> BOOLEAN b:          $(SMXflow)
VFUN FCAterminalPathSet(seid terminalGroup) -> SET_OF seid ss;
VFUN FCAcurrentPath(seid terminalGroup) -> seid s;
    $(Note that the parameter types of the two FCA VFUNS has been changed
      from a separate DESIGNATOR type to a seid.
     These changes must also be made in the kernel specs.)
VFUN SENseidNsp(seid anySeid) -> INTEGER nsp;


    ASSERTIONS

TRUE;   $(Placeholder for the ASSERTIONS paragraph.)


    FUNCTIONS

$(--------------------VFUNS-------------------------)

$("The important VFUNS are:
UAADB (User Authentication Access Data Base)
TPDB (Terminal Profile Data Base)
SDB (System Data Base)
GADB (Group Authorization Data Base)
FCAterminalPathSet
FCAcurrentPath
terminalUser (for a given terminal, tells the user, as given by
    the most recent Login)
ptbl (a table showing, for a given process seid, the terminal
    and terminal path in use by that seid)
auditTrail (for sending audit messages)
securePath   (path used when ATTN key is struck on a terminal)


")

VFUN UAADB(VECTOR_OF CHAR userName) -> STRUCT_OF(
    VECTOR_OF CHAR passWord;
    INTEGER consecutiveUnsuccessfulLogins;
    BOOLEAN loginsPermitted;
    seid initialProcess;
    tiiStruct initialLevel;
    nonDisType maxLevel)  ans;
       $("Life cycle of UAADB:  logIn reads the following:
       passWord, loginsPermitted, initialProcess,  initialLevel, maxLevel ;
       logIn updates consecutiveUnsuccessfulLogins")
    HIDDEN;
    INITIALLY
      TRUE;
    $(***Perhaps there are some initial restrictions?)

VFUN TPDB(seid terminalGroup ) -> STRUCT_OF(
    BOOLEAN locked,
            configured,
            loggedIn;
    nonDisType maxLevel) ans :

       $(Life cycle of TPDB: logIn reads the following fields:
       locked, configured, loggedIn, maxLevel.
       changeAccessLevel reads maxLevel;  logIn updates loggedIn.)

    HIDDEN;
    INITIALLY
      TRUE;

VFUN SDB() -> STRUCT_OF(
    nonDisType maxLevel) x; $(given as STRUCT to allow for more fields later)
    $(read by changeAccessLevel, logIn, fileAccessModifier)
    HIDDEN;
    INITIALLY
      TRUE;

VFUN GADB(INTEGER x) -> STRUCT_OF(
      nonDisType maxLevel;
      VECTOR_OF CHAR password) y ;
        $(read by changeGroup, changeAccessLevel, fileAccessModifier)
    HIDDEN;
    INITIALLY
      TRUE;

VFUN terminalUser(seid terminalGroup) -> VECTOR_OF CHAR user:
    $(initialized by logIn; finalized by logOut; read by fileAccessModifier)
    HIDDEN;
    INITIALLY
    FORALL seid x : terminalUser(x) =?:


VFUN ptbl(seid pSeid) -> STRUCT_OF(
    seid t,    $("terminal")
          p) x ;   $("path used by pSeid on terminal t")
    HIDDEN:
    INITIALLY
      FORALL seid x: ptbl(x) =? ;
    $("ptbl is initialized by the kernel.  For simplicity in these
     specs, however, we present only the above initialization.
     ptbl is updated by logIn; read by logOut; and finalized by
     logOut; In the implementation there is the additional complexity
     that the logIn process might create new processes, which are
     also added to ptbl. We ignore this complexity in these specs")

VFUN auditTrail() -> VECTOR_OF VECTOR_OF CHAR x;
    HIDDEN;
    INITIALLY
       LENGTH(x) =0;
  $("auditTrail is a sequence of messages sent to the audit capture process.
   In the implementation, each message is fixed-length and of a given
   format. Such restrictions are ignored here; the main purpose of
   including auditTrail in these specs is to show when such messages
   are sent")

VFUN securePath(seid terminalGroup) -> seid x;
     HIDDEN;
     INITIALLY
        x INSET FCAterminalPathSet(terminalGroup);

VFUN charsToInt(VECTOR_OF CHAR x) -> INTEGER y;
     HIDDEN;
     INITIALLY
         TRUE;
  $(A minor conversion function  used by fileAccessModifier)



$(----------------OFUNS-----------------------)

$("SecureInitiator and SecureServer are not given explicitly
in these specs. Instead, the main events they handle,
namely, detecting that the attention key is struck, detecting
that a hangup has occurred destroying the physical connection between
a terminal and the cpu, and operator requested logout -- are given
by the OFUNS ATTN, HANGUP, and OPREQLOGOUT. The remainder of the
functionality of the SecureServer is that of a command interpreter.
In line with the HDM model of computation, the semantics for the
individual commands is given below, but the notion of an explicit
command interpreter is not given.")

OFUN ATTN()[seid terminalGroup];
  $(In response to the attention key being struck on the terminal)

    EFFECTS

  'FCAcurrentPath(terminalGroup) = securePath(terminalGroup);
    $("Since TIIinfo(securePath(t)) never changes, there is no need
     to set it here.")


OFUN HANGUP()[seid terminalGroup];
  $(In response to the detection of a broken terminal connection)

        DEFINITIONS
   seid t IS terminalGroup;

        EXCEPTIONS
 notLoggedIn:  NOT TPDB(t).loggedIn;

        EFFECTS
 FORALL seid x | ptbl(x).t = t AND
   ptbl(x).p ~= securePath(t)  :
        'TIIinfo(x) = ? AND 'ptbl(x) =?;
        $(Wipe out all processes which are using a non-secure path on the
        effected terminal.)

 'terminalUser(t) = ?;
 'TPDB(t).loggedIn = FALSE;
 FORALL seid x INSET FCAterminalPathSet(t):
   x ~= securePath(t) =>
        'TIIinfo(x) =?;
 'FCAcurrentPath(t) = ?;
   $(Wipe out all paths, except secure path, on the effected terminal.)

OFUN OPREQLOGOUT(seid terminalGroup)[seid pSeid]:
 $(Note that terminalGroup is given explicitly for an operator
   requested logout, in contrast to normal logout or hangup)

      DEFINITIONS
 seid t IS terminalGroup;

      EXCEPTIONS
 notLoggedIn:  NOT TPDB(t).loggedIn;
 badpSeid:  TIIinfo(pSeid).nd ~= STRUCT(sysHighLevel, allSecCats, sysHighLevel,
                                        allIntCats);
 $(badpSeid indicates that somebody other than the operator is attempting
   to invoke OPREQLOGOUT)

      EFFECTS
 FORALL seid x | ptbl(x).t = t AND
   ptbl(x).p ~= securePath(t)   :
        'TIIinfo(x) = ? AND 'ptbl(x) =?;

 'terminalUser(t) = ?;
 'TPDB(t).loggedIn = FALSE;
 FORALL seid x INSET FCAterminalPathSet(t):
   x ~= securePath(t)  =>
        'TIIinfo(x) =?;
 'FCAcurrentPath(t) = ?;
 'auditTrail() = VECTOR (FOR i FROM 1 TO LENGTH(auditTrail())+1:
        IF i <= LENGTH(auditTrail()) THEN auditTrail()[i]
        ELSE "operator requested logout" );


OFUN logIn(VECTOR_OF CHAR userName,passWord) [seid terminalGroup];

      DEFINITIONS
 seid t IS terminalGroup;
 seid p IS SOME seid q | q INSET FCAterminalPathSet(t) AND
   TIIinfo(p)=?;
```

```
  $("Note that p cannot be securePath(t)")
tiiStruct s IS UAADB(userName).initialLevel;
nonDisType ndLevel IS minimum({
  s.nd,
  TPDB(t).maxLevel,
  UAADB(userName).maxLevel,
  GADB(UAADB(userName).initialLevel.group).maxLevel,
  SDB().maxLevel});
tiiStruct startLevel IS STRUCT(ndLevel, s.da, s.owner, s.group, s.priv);

  EXCEPTIONS
badt: TPDB(t) = ?;
badTPDBdata: TPDB(t).locked OR NOT TPDB(t).configured
   OR TPDB(t).loggedIn  OR NOT UAADB(userName).loginsPermitted;
undefUser:  UAADB(userName) =?;
noPath: p = ?;
badLevel: ndLevel =?;       $(This should never happen)

  EFFECTS
IF (passWord ~= UAADB(userName).passWord)    THEN
  'UAADB(userName) = STRUCT(UAADB(userName).passWord,
      UAADB(userName).consecutiveUnsuccessfulLogins +1,
      UAADB(userName).loginsPermitted,
      UAADB(userName).initialProcess,
      UAADB(userName).initialLevel,
      UAADB(userName).maxLevel)
ELSE
  'UAADB(userName) = STRUCT(UAADB(userName).passWord,
      0,
      UAADB(userName).loginsPermitted,
      UAADB(userName).initialProcess,
      UAADB(userName).initialLevel,
      UAADB(userName).maxLevel)                             AND
  'TIIinfo(p) = startLevel                                  AND
  'TIIinfo(UAADB(userName).initialProcess) = startLevel     AND
  'terminalUser(t) = userName                               AND
  'TPDB(t).loggedIn = TRUE                                  AND
  'FCAcurrentPath(t) = p                                    AND
  'ptbl(UAADB(userName).initialProcess) = STRUCT(t,p)  ;
'auditTrail() = VECTOR (FOR i FROM 1 TO LENGTH(auditTrail())+1:
      IF i <= LENGTH(auditTrail()) THEN auditTrail()[i]
      ELSE "login attempted" );



OFUN logOut()[seid terminalGroup];

     DEFINITIONS
 seid t IS terminalGroup;

     EXCEPTIONS
 notLoggedIn:  NOT TPDB(t).loggedIn;

     EFFECTS
 FORALL seid x | ptbl(x).t = t AND
   ptbl(x).p ~= securePath(t)  :
        'TIIinfo(x) = ? AND 'ptbl(x) =?;
  'terminalUser(t) = ?;
  'TPDB(t).loggedIn = FALSE;
  FORALL seid x INSET FCAterminalPathSet(t):
    x ~= securePath(t)  =>
        'TIIinfo(x) =?;

  'FCAcurrentPath(t) =?;
  'auditTrail() = VECTOR (FOR i FROM 1 TO LENGTH(auditTrail())+1:
        IF i <= LENGTH(auditTrail()) THEN auditTrail()[i]
        ELSE "logout" );


OFUN changeAccessLevel(seid terminalGroup; tiiStruct desiredLevel)  ;

      DEFINITIONS
  seid t IS terminalGroup;
  seid q IS SOME seid p |
    p INSET FCAterminalPathSet(t) DIFF {securePath(t)} AND
    TIIinfo(p).nd = desiredLevel.nd                    AND
    TIIinfo(p).owner = desiredLevel.owner              AND
    TIIinfo(p).group = desiredLevel.group ;
  seid r IS SOME seid p |
    p INSET FCAterminalPathSet(t) DIFF {securePath(t)} AND
    TIIinfo(p) = ?;
  nonDisType ndLevel IS minimum({
     TPDB(t).maxLevel,
     SDB().maxLevel,
     UAADB(terminalUser(t)).maxLevel,
     GADB(TIIinfo(FCAcurrentPath(t)).group).maxLevel
     });
  tiiStruct cl $(current level) IS
        TIIinfo(FCAcurrentPath(t));
  tiiStruct startLevel IS STRUCT(ndLevel,cl.da,cl.owner,cl.group,cl.priv);

      EXCEPTIONS
  noPath: q = ? AND r = ?;
  badLevel:  NOT LE(desiredLevel.nd, ndLevel);

      EFFECTS
  IF q~= ? THEN
     'FCAcurrentPath(t) = q
  ELSE
     'TIIinfo(r) = startLevel                                    AND
     'TIIinfo(UAADB(terminalUser(t)).initialProcess) = startLevel  AND
     'FCAcurrentPath(t) = r ;
  'auditTrail() = VECTOR (FOR i FROM 1 TO LENGTH(auditTrail())+1:
        IF i <= LENGTH(auditTrail()) THEN auditTrail()[i]
        ELSE "change access level" );

OFUN changeGroup(seid terminalGroup; seid desiredGroup;
    VECTOR_OF CHAR pwdDesiredGroup) [seid pSeid] ;
   $(pSeid is the seid in whose behalf the change is being made)

      DEFINITIONS
  seid t IS terminalGroup;
  tiiStruct ptii IS TIIinfo(pSeid);
  seid q IS SOME seid p |
    p INSET FCAterminalPathSet(t) DIFF {securePath(t)} AND
    TIIinfo(p).nd = TIIinfo(desiredGroup).nd                    AND
    TIIinfo(p).owner = TIIinfo(desiredGroup).owner              AND
    TIIinfo(p).group = TIIinfo(desiredGroup).group ;
  seid r IS SOME seid p |
    p INSET FCAterminalPathSet(t) DIFF {securePath(t)} AND
    TIIinfo(p) = ?;
  nonDisType ndLevel IS minimum({
    TPDB(t).maxLevel,
    SDB().maxLevel,
    UAADB(terminalUser(t)).maxLevel,
    TIIinfo(FCAcurrentPath(t)).nd
    });
  tiiStruct cl $(current level) IS
        TIIinfo(FCAcurrentPath(t));
  tiiStruct startLevel IS STRUCT(cl.nd,cl.da,cl.owner,
                 SENseidNsp(desiredGroup),
                 cl.priv):

      FXCEPTIONS
  badt:  TPDB(t) =?;
  noPath: q = ? AND r = ?:
  badLevel:  NOT LE(GADB(SENseidNsp(desiredGroup)).maxLevel,ndLevel);
  badGroup:  GADB(SENseidNsp(desiredGroup)) = ?;
  badPassWord:  pwdDesiredGroup ~=
              GADB(SENseidNsp(desiredGroup)).password;

      EFFECTS
  'TIIinfo(pSeid) = startLevel;
  IF q~= ? THEN
     'FCAcurrentPath(t) = q
  ELSE
     'TIIinfo(r) = startLevel                                    AND
     'TIIinfo(UAADB(terminalUser(t)).initialProcess) = startLevel  AND
     'FCAcurrentPath(t) = r ;
  'auditTrail() = VECTOR (FOR i FROM 1 TO LENGTH(auditTrail())+1:
        IF i <= LENGTH(auditTrail()) THEN auditTrail()[i]
        ELSE "change group" );


OFUN fileAccessModifier(seid terminalGroup, fSeid; nonDisType desirednd;
   modeStruct da);

      DEFINITIONS
  seid t IS terminalGroup;
  tiiStruct ftii IS TIIinfo(fSeid);

      EXCEPTIONS
  badt: TPDB(t) = ?;
  badFile: TIIinfo(fSeid) = ?:
  badOwner: ftii.owner ~= charsToInt(terminalUser(t));
    $(User must own file he is trying to modify)
  badFileLevel: ftii.nd ~=
    minimum(
      ftii.nd,
      TPDB(t).maxLevel,
      SDB().maxLevel,
      UAADB(terminalUser(t)).maxLevel,
      GADB(UAADB(terminalUser(t)).initialLevel.group).maxLevel
    });
  badDesiredLevel:  desirednd ~=
    minimum({
      desirednd,
      TPDB(t).maxLevel,
      SDB().maxLevel,
      UAADB(terminalUser(t)).maxLevel,
      GADB(UAADB(terminalUser(t)).initialLevel.group).maxLevel
    });


      EFFECTS
  'TIIinfo(fSeid) = STRUCT(desirednd, da, ftii.owner, ftii.group,
     ftii.priv);
  'auditTrail() = VECTOR (FOR i FROM 1 TO LENGTH(auditTrail())+1:
        IF i <= LENGTH(auditTrail()) THEN auditTrail()[i]
        ELSE "file fSeid modified to desired nd" );


END_MODULE
```

```
MODULE alc         $(access level control)
$("
COPYRIGHT:  1978, by Ford Aerospace and Communications Corp.
ADDRESS:    Ford Aerospace and Communications Corp.
            Western Development Laboratories
            3939 Fabian Way
            Palo Alto, California 94303
            Attention: Software Technology Dept.
            Mail Stop: V02

CPCI:  NKSR
CPC:

MODULE NAME:  alc  access level control
FUNCTION:
ASSUMPTIONS:
HISTORY:
  AUTHOR:   Tom Berson
  VERSION:  1.1 of 10/18/78
  MODULE TYPE: SPECIAL Specifications
SPECIAL NOTES:
")


TYPES

accessLevel: STRUCT_OF(INTEGER securityLevel,integrityLevel;
                     SET_OF securityCat sCatSet);

PARAMETERS

accessLevel ALCgroundLevel; $(level for terminals not logged in)
accessLevel ALCloginLevel;  $(initial level for all logins)


EXTERNALREFS

        FROM smx:
securityCat: DESIGNATOR:
integrityCat: DESIGNATOR;


FUNCTIONS

VFUN ALCsetPermissible(accessLevel p,max)->BOOLEAN B;

        DERIVATION
    (p.securityLevel<=max.securityLevel)
AND (p.sCatSet SUBSET max.sCatSet)
AND (p.integrityLevel<=max.integrityLevel);


END_MODULE
```

```
$("   MODULE:         despooler.specs  (Version 1.9, 14:13:09)
                      Module Name (dsp.specs)
                      File Name (/kr/sunix/specs/nksr/s.dsp.specs)
      CONTENTS        despooler operations
      TYPE:           SPECIAL specifications
      LAST CHANGED:   11/21/80,14:13:09

  ")

MODULE despooler

$("The main purpose of the despooler is to periodically check
if files are spooled for printing, and if so, queue
them to be printed one at a time.  One main interface is thus
to the spooler mechanism, which indicates the currently spooled
files in the VFUN spooledFiles.  The spooler mechanism puts files
from the "outside world" into spooledFiles via the OFUN spool.

The other main interface of the despooler is to the "operator",
i.e., someone who sends the following sorts of commands to the
despooler:  start, stop, die, setLevel.  Start, stop, and die
force the despooler to look like a state transition machine, in that
printing of files can only occur from state started.  The command
setLevel does not affect the state transition aspects, but merely
changes the set of files that will actually be printed.
Also, the command start can optionally specify a new level at which
the despooler will be started.  In these specs, the new level will
always be given as a parameter of start.  The level
of each file to be printed must be <= maxPrintLevel, which is a VFUN
modified as a result of interpreting the command setLevel or start.  For each
of the four commands, the operator also sends an indication of whether
the command should be interpreted "immediately" or after the currently
queued files are all printed.

    In addition to the "applications-oriented" tasks of queueing and
printing files, the interrupt-handling aspects of the operator commands
must also be indicated in these specs.

    As a way of structuring these specs, the following six categories
of objects are useful:

  I.  Operator Commands - setLevel, start, stop, die.
  II. Interrupt Buffers - Boolean VFUNS which are initially FALSE;
        they are set to TRUE by OFUN issueCommand, and reset to FALSE
        when the interrupt is actually processed by OFUN processInterrupts.
        The Boolean VFUNS are: levbuf, startbuf, stopbuf, and diebuf,
        corresponding respectively to the four operator commands above.
        In addition, there is the buffer newLevbuf, indicating the new level
        for a setLevel or start command.
  III. ILPL events.  Below are given abstract programs in ILPL for
        the spooler, despooler, and operator.  One of the structuring
        mechanisms of ILPL is based on Zahn's "event-indicator" construct
        (see Knuth's "Str. Pgmg with Gotos" article).
        The ILPL events used in the programs below are:
        startE,stopE,dieE, immInt ("immediate interrupt" ), Qempty.
  IV.  BOOLEAN  VFUNS corresponding to the ILPL events.
```

```
        VstartE, VstopE, VdieE.  Initially FALSE;  tested by the ILPL
        programs to determine if the corresponding events should be
        signalled.  Set to TRUE by OFUN processInterrupts.  Reset
        to FALSE by OFUNS given in the next category, which are invoked
        by the ILPL program prior to signalling the event.
        (No specific VFUNS are needed for Qempty, which gets signalled
        when LENGTH(queue) = 0; or for immInt, which gets signalled
        when nowbuf is TRUE.
  V.   OFUNS to reset the BOOLEAN VFUNS in category IV to FALSE:
        OstartE, OstopE, OdieE.
        These OFUNS are invoked by the ILPL program, and reset the
        corresponding VFUNS to FALSE.
  VI.  OFUNS and VFUNS appropriate to spooler/despooler functionality.
        The following are the major ones:
        VFUNS:  spooledFiles - spooler puts files here;  despooler
                        takes files from here and puts them on queue.
                queue - files wait here until they are printed.
                        If a setlevel interrupt occurs, the queue may
                        be cleared prior to printing the files.  The
                        files are still in spooledFiles.
                output - simulates line printer output.  New printed
                        pages are added to the right of this vector.
                maxPrintLevel - changed by setlevel.  Only files <=
                        to this level may be moved from the outside
                        world to spooledFiles, or from spooledFiles
                        to queue.
                currentState - handles state transitions;  used mainly
                        by OFUN processInterrupts.

        OFUNS:
                issueCommand - receive command from operator, and set
                        interrupt buffers appropriately.
                processInterrupts - check the interrupt buffers
                        and take appropriate action.  Handle state
                        transitions.
                processQelement - dequeue and print the file at the
                        beginning of the queue.
                fillQueue - move files from spooleFiles to queue.
                spool - move files from the outside world to spooledFiles.

----------------------------------------------------------------------------

The ILPL program for the despooler is given next.

(cycle until a start command is received.  This also sets maxPrintLevel.)
UNTIL dieE DO
  UNTIL stopE,dieE DO
    processInterrupts;
    IF VstopE THEN
      OstopE;
      SIGNAL(stopE);
    END_IF
    IF VdieE THEN
      OdieE;
      SIGNAL(dieE);
    END_IF
    fillQueue;
    UNTIL Qempty, immInt DO
      IF LENGTH(queue) = 0 THEN;
        SIGNAL(Qempty)
      END_IF
      IF nowbuf() THEN;
        SIGNAL(immInt);
      END_IF
      processQelement;
    THEN
      ON Qempty: ;
      ON immInt:
        processInterrupts;
        IF VstopE THEN
          OstopE;
          SIGNAL(stopE);
        END_IF
        IF VdieE THEN
          OdieE;
          SIGNAL(dieE);
        END_IF
    END
  THEN
    ON dieE: SIGNAL(dieE);
    ON stopE:
      UNTIL startE DO
        processInterrupts;
        IF VstartE() THEN
          OstartE;
          SIGNAL(startE);
        END_IF;
      THEN
        ON startE:  ;
      END;
  END;
THEN
ON dieE:  "abort";
END
---------------------------------------------------------------------
The abstract program for the spooler is:

UNTIL doomsday DO
  spool(cFile,tFile);
END;
---------------------------------------------------------------------
The abstract program for the operator is:

UNTIL doomsday DO
  issueCommand(comd);
END;


The operator is "supposed" to issue commands in accordance with
the following state transition diagram.

             setlevel (level)
```

```
           |---------|
           |         V          start (level)
       *---------------* <<-------------------- *-------*
       |started (level)|                        |stopped |
       *---------------* -------------------->> *-------*
                     \          stop                  *
                      \                               *
                       \         *----*               |
                        \------>>|dead|               |
                           die   *----*               |
                                   |                  |
                                   |--------------->>>
                                     (system startup)

processInterrupts makes sure that this diagram is obeyed.
--------------------------------------------------------------------------

")


    TYPES
$(from smx)
nonDisType: STRUCT_OF(
                INTEGER securityLevel; SET_OF securityCat securityCatS;
                INTEGER integrityLevel; SET_OF integrityCat integrityCatS);
  $(integrityCat is typically the null set)
daType: SET_OF daMode;
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
tiiStruct: STRUCT_OF(nonDisType nd;
                modeStruct da; INTEGER owner, group; SET_OF privType priv);
    $(from fca)
globalData: STRUCT_OF(INTEGER linkCount, openCount, timeLastMod; seid subtype;
                       BOOLEAN openAtCrash);
  $(state information for all openable objects)


$(Types for despooler are given next.)
line:  {VECTOR_OF CHAR x | LENGTH(x) <= maxLineLength};
pageImage:  {VECTOR_OF line x| LENGTH(x) <= maxPageLength};
outStream:  VECTOR_OF pageImage;
textImage:  VECTOR_OF line;
     $(variables of type textImage will be text files containing
       an unbounded but finite sequence of lines.)
commands:  {setLevel,stop,die,start};
        $(The commands are IPC messages that can be received by
        the despooler.)
dspStates:  {started,stopped,dead};
marMsgType :{VECTOR_OF line x | LENGTH(x) = marginHeadingLength};
  $(type of the security message that appears at top and bottom of
  each output page.)

    PARAMETERS

INTEGER maxPageLength $(Number of physical lines on a physical page) ;
INTEGER maxLineLength;
INTEGER marginHeadingLength; $(Number of lines in top margin heading
                  that appears on each output page;  this same number
                  of lines also appears at the bottom margin of
                  each output page.)
marMsgType marginMsg(seid cFile);
                  $(Message appearing at top and bottom of each printed
                  physical page, giving owner, security level, etc.)
nonDisType nullLevel;  $(when die, set maxPrintLevel to nullLevel)
VECTOR_OF seid emptyQueue;  $(Set queue to emptyQueue when level changes)

    DEFINITIONS

INTEGER maxNoTextLines IS
  maxPageLength - 2*marginHeadingLength;
        $(max number of usable lines per output page.)
        $(margin message appears at top and bottom of page.)

INTEGER numPages(textImage file) IS
  LENGTH(file)/maxNoTextLines +
     (IF LENGTH(file) MOD maxNoTextLines =0 THEN 0 ELSE 1);
     $(Tells how many output pages a file will need to be printed.)

pageImage bannerPage(line msg) IS
  VECTOR(FOR i FROM 1 TO maxPageLength:
    IF i <= marginHeadingLength THEN
       marginMsg[i]
    ELSE IF i = marginHeadingLength + 1 THEN
       msg
    ELSE IF i INSET {marginHeadingLength+2 .. maxPageLength
           - marginHeadingLength } THEN
       " "
    ELSE marginMsg[i - maxPageLength + marginHeadingLength]
      $(Put margin message at bottom of page.)
    )
    ;

seid seidAssoc(nonDisType x) IS
  SOME seid y| TIIinfo(y).nd = x;  $(need a seid, rather than nonDisType,
                                   to pass as a parameter to SMXflow.)

    EXTERNALREFS

    FROM smx:
INTEGER SENlowLevel;
INTEGER SENhighLevel;
seid: DESIGNATOR;
secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
                   tExtent, tNull};
privType: {
        privFileUpdateStatus,   privLink,       privLockSeg,
        privModifyPriv,         privMount,
        privSetFileLevel,       privSetSegProcLevel,
        privStickySeg,          privTerminalLock,
        privViolSimpSecurity,   privViolStarSecurity,
        privViolSimpIntegrity,  privViolStarIntegrity,
        privViolDiscrAccess,    privSignal,     privWalkPTable,
        privHalt,               privKernelCall, privViolCompartments,
          privRealizeExecPermissions};

daMode: {daRead, daWrite, daExecute};
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;
domainType: {userDomain, supervisorDomain};

VFUN TIIinfo(seid s) -> tiiStruct st;
VFUN SMXflow(seid pSeid, oSeid; daType da) -> BOOLEAN b;           $(SMXflow)
VFUN SMXdap(seid pSeid, oSeid; daType da) -> BOOLEAN b;
VFUN SENseidNsp(seid anySeid) -> INTEGER nsp;
VFUN SENseidToInt(seid anySeid) -> INTEGER i;
VFUN SENseidType(seid s) -> secureEntityType set;              $(SENseidType)
VFUN SMXhasPriv(seid pSeid; privType privl) -> BOOLEAN b;        $(SMXhasPriv)

      FROM fca:
VFUN FCAinfo(seid fSeid) -> globalData gl;                         $(FCAinfo)

    FUNCTIONS

VFUN spooledFiles() -> SET_OF seid controlFiles;
  $(Assumed to be set by spooler;  for simplicity in specs,
  will only contain control files, rather than <control file,
  text file> pairs.  A separate VFUN, cToTfile, will give the
  text file corresponding to any control file in spooledFiles.)

  HIDDEN;
  INITIALLY controlFiles = {};

VFUN cToTfile(seid controlFile) -> seid textFile;
  $(Assumed to be set by spooler;  contains the text file associated
  with a given control file, which is represented in the implementation
  as a set of <control file, text file> pairs.)

  HIDDEN;
  INITIALLY textFile  = ?;

VFUN copies(seid cFile) -> INTEGER x;
  $(Assumed set by spooler.  Gets control file info; specifically,
  how many copies of the file to print.)

  HIDDEN;
  INITIALLY x =?;

VFUN textOfSeid(seid x) -> textImage y;

  HIDDEN;
  INITIALLY y = ?;
  $(Assumed set by spooler.)
  $(An abstraction away from the representation via the directory structure.)

VFUN currentState() -> dspStates x;

  HIDDEN;
  INITIALLY x = stopped;  $(Requires an interrupt to get started.)

```

```
VFUN queue() -> VECTOR_OF seid qe;
   $(One of the main VFUNS in despooler.  Files are copied from
   spooledFiles to queue by the OFUN fillQueue.  From queue, the
   files are printed one at a time.  Files remain on both spooledFiles and
   queue until the files are printed, or determined to be unprintable.
   When printed, they are removed from both spooledFiles and queue.
   )

   INITIALLY qe =?;


VFUN maxPrintLevel() -> nonDisType x;
   $("Only files at or below maxPrintLevel may be printed.  This level
   is set, in the implementation, at the beginning of the despooler code.
   It is set in these specs via the OFUN initPrLevel.
   It may be changed via the IPC command setLevel.")

   $("In the implementation, there are two separate "filters"
   appropriate in file movement.  active.maxLevel is an upper bound
   on security level for files moving from the outside world to spooledFiles,
   and maxPrintLevel is an upper bound on files moving from spooledFiles
   to queue.  In the specs, for simplicity, the same filter, maxPrintLevel,
   is used in both circumstances.")



   HIDDEN;
   INITIALLY x=?;


VFUN output() -> outStream x;
   $(output is a write-only VFUN simulating actual page printing
   by appending new pages to the right end of of the output vector.)


   INITIALLY LENGTH(x) = 0;

$(The following VFUNS are the interrupt buffers.)

VFUN nowbuf() -> BOOLEAN x;
   INITIALLY x = FALSE;

VFUN levbuf() -> BOOLEAN x;
   HIDDEN:
   INITIALLY x = FALSE;

VFUN startbuf() -> BOOLEAN x;
   HIDDEN;
   INITIALLY x = FALSE;

VFUN stopbuf() -> BOOLEAN x;
   HIDDEN;
   INITIALLY x = FALSE;

VFUN diebuf() -> BOOLEAN x;
   HIDDEN:
```

```
  INITIALLY x = FALSE;

VFUN newLevbuf() -> nonDisType x;
  HIDDEN;
  INITIALLY x = ?;

$(The following VFUNS correspond to ILPL events.)

VFUN VstartE() -> BOOLEAN x;
  INITIALLY x = FALSE;

VFUN VstopE() -> BOOLEAN x;
  INITIALLY x = FALSE;

VFUN VdieE() -> BOOLEAN x;
  INITIALLY x = FALSE;


$(-----------------------OFUNS-------------------------------------- )

OFUN processInterrupts();
  $("Check interrupt buffers and take appropriate action.  Interrupts
   must be in accordance with above state transition diagram.")

  EXCEPTIONS
    badTrans1:  currentState() = started AND startbuf();
    badTrans2:  currentState() = dead;
    badTrans3:  currentState() = stopped AND
                       (stopbuf() OR diebuf() OR levbuf());
    $(The above exceptions rule out illegal commands from each state.
     Handling of multiple legal interrupts from state started is as
     follows.  Set level and stop occuring as multiple interrupts
     effectively stop the abstract machine at the new level.  Set level
     and die, but not stop, occuring as multiple interrupts essentially
     die at the null level, with the queue cleared.  Stop and die
     occuring as multiple interrupts effectively treat the interrupts
     sequentially, first processing the stop.  Set level, stop, and
     die occuring together effectively group set level and stop
     together as described above, and then sequentially handle the
     die.  "Sequentially" is used here wrt  consecutive calls
     of processInterrupts at the ILPL level.

     This treatment of interrupts is an abstraction and simplification
     vis-a-vis the implementation, which queues the interrupts on the
     same queue that files reside, but is logically equivalent to the
     implementation.
     )


  EFFECTS
    IF levbuf() AND stopbuf() THEN
      'output() = VECTOR(FOR i FROM 1 TO LENGTH(output())+1:
         IF i <= LENGTH(output()) THEN output()[i]
         ELSE bannerPage("new level and stop"))  AND
      'levbuf() = FALSE   AND
      'currentState() = stopped  AND
      ´maxPrintLevel() = newLevbuf() AND
      ´queue() = emptyQueue AND
      ´VstopE() = TRUE AND
      ´stopbuf() = FALSE

    ELSE IF levbuf() AND diebuf() THEN
      ´output() = VECTOR(FOR i FROM 1 TO LENGTH(output())+1:
         IF i <= LENGTH(output()) THEN output()[i]
         ELSE bannerPage("new level and die"))  AND
      ´levbuf() = FALSE   AND
      ´currentState() = dead  AND
      ´maxPrintLevel() = nullLevel  AND
      ´queue() = emptyQueue AND
      ´VdieE() = TRUE AND
      ´diebuf() = FALSE

    ELSE IF levbuf() THEN
      ´maxPrintLevel() = newLevbuf()    AND
        ´output() = VECTOR(FOR i FROM 1 TO LENGTH(output())+1:
           IF i <= LENGTH(output()) THEN output()[i]
           ELSE bannerPage("new level")
           )    AND
         ´levbuf() = FALSE  AND
         ´queue() = emptyQueue

    ELSE IF stopbuf() THEN
      ´currentState() = stopped  AND
      ´VstopE() = TRUE AND
      ´stopbuf() = FALSE AND
      ´output() = VECTOR(FOR i FROM 1 TO LENGTH(output())+1:
         IF i <= LENGTH(output()) THEN output()[i]
         ELSE bannerPage("stopped")
         )

    ELSE IF diebuf() THEN
      ´currentState() = dead  AND
      ´maxPrintLevel() = nullLevel AND
      ´VdieE() = TRUE AND
      ´diebuf() = FALSE AND
      ´output() = VECTOR(FOR i FROM 1 TO LENGTH(output())+1:
         IF i <= LENGTH(output()) THEN output()[i]
         ELSE bannerPage("dead")
         )

    ELSE IF startbuf() THEN
      ´currentState() = started  AND
      ´maxPrintLevel() = newLevbuf() AND
      ´queue() = emptyQueue AND
      ´VstartE() = TRUE AND
      ´startbuf() = FALSE AND
      ´output() = VECTOR(FOR i FROM 1 TO LENGTH(output())+1:
         IF i <= LENGTH(output()) THEN output()[i]
         ELSE bannerPage("started")
         )
    ELSE TRUE
   ;
```

```


OFUN issueCommand(commands cmd; nonDisType level; BOOLEAN imm);
  $(Set the appropriate interrupt buffers based on the command.)

  $(****Put in exception check that issueCommand is invoked by
   someone at or above operator integrity.)


 EFFECTS
  IF cmd = setLevel THEN
    'levbuf() = TRUE  AND
    'newLevbuf() = level
  ELSE IF cmd = start THEN
    'startbuf() = TRUE   AND
    'newLevbuf() = level
  ELSE IF cmd = stop THEN
    'stopbuf() = TRUE
  ELSE IF cmd = die THEN
    'diebuf() = TRUE
  ELSE
    TRUE;

  IF imm THEN
    'nowbuf() = TRUE
  ELSE
    TRUE;

OFUN spool(seid cFile,tFile: INTEGER ncopies: textImage text);
  $(Move the file from the "outside world" to spooledFiles.  File
   must be at or below maxPrintLevel.  In the implementation,
   active.maxLevel is used instead of maxPrintLevel in moving files
   to spooledFiles, and maxPrintLevel is used in moving files from
   spooledFiles to queue.  For simplicity in these specs, the same
   discriminator, namely maxPrintLevel, is used  for both.)

  EXCEPTIONS
    badLevel:  NOT SMXflow(cFile, seidAssoc(maxPrintLevel()), {daWrite});

  EFFECTS
    'spooledFiles() = spooledFiles() UNION {cFile};
    'cToTfile(cFile) = tFile;
    'copies(cFile) = ncopies;
    'textOfSeid(tFile) = text;


OFUN fillQueue();
  $("Copy files from spooledFiles to queue, without currently removing
  them from spooledFiles.  Only files whose current level is less
  than or equal maxPrintLevel are copied.  ")

  DEFINITIONS

    SET_OF seid queueableFiles IS  {seid x | x INSET spooledFiles() AND
          $(1. Not in queue) (FORALL INTEGER j INSET {1 .. LENGTH(queue())} :
              queue()[j] ~= x)
```

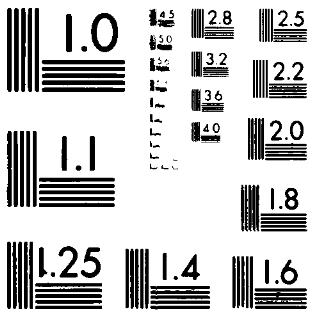
```
      AND $(2. ) SMXflow(x, seidAssoc(maxPrintLevel()), {daWrite})
      }
    ; $(queueableFiles is the set of files about to move onto queue)

    VECTOR_OF seid setToVec(SET_OF seid x) IS
      SOME_VECTOR_OF seid y | LENGTH(y) = CARDINALITY(x)
        AND (FORALL INTEGER i INSET {1 .. CARDINALITY(x)}:
          y[i] INSET x)
        AND  (FORALL seid j INSET x:  EXISTS INTEGER k INSET
          {1 .. CARDINALITY (x)}: j = y[k])
    ; $(setToVec merely "linearizes" a set into some arbitrary order.
       In the implementation, the order in which files are queued
       is based partly on file length.  In these specs, the order is arb.)


  EXCEPTIONS

    XnoObj:  EXISTS seid x INSET spooledFiles():
        TIIinfo(cToTfile(x)) =?;
    XnoFile:  EXISTS seid x INSET spooledFiles():
          FCAinfo(cToTfile(x))  =?;


  ASSERTIONS
    LENGTH(queue()) = 0;
      $(" Whenever fillQueue is invoked the queue is empty because.
      queue starts out empty;  after queue is filled by fillQueue,
      each element on the queue is processed prior to fillQueue being
      reinvoked, or else an interrupt occurs forcing queue to be cleared
      prior to fillQueue being reinvoked.")


  EFFECTS

      'queue() = VECTOR(FOR i FROM 1 TO CARDINALITY(queueableFiles):
          setToVec(queueableFiles)[i]) ;


OFUN processQelement();
  $(dequeue the first element of queue and print associated file.)

  DEFINITIONS
    seid cFileSeid IS queue()[1];
    textImage text IS textOfSeid(cToTfile(cFileSeid));
    INTEGER curLength IS numPages(text) + 2;  $(2 extra: banner & trailer)

  EXCEPTIONS
    emptyQ:  LENGTH(queue()) = 0;

  EFFECTS
    'output() = VECTOR(FOR i FROM  1 TO LENGTH(output()) +
              copies(cFileSeid)*(numPages(text)+2):
                                         $(banner + trailer page
                                         for each copy of the file)
        IF i <= LENGTH(output()) THEN output()[i]
        ELSE IF (i - LENGTH(output())) MOD curLength INSET {0,1}  THEN
```

```
              bannerPage("normal file")
            ELSE IF (i - LENGTH(output())) MOD curLength = curLength -1 THEN
                $(last page of text may not be full)
              VECTOR(FOR j FROM 1 TO LENGTH(text) MOD maxNoTextLines:
                  text[LENGTH(text)  - (LENGTH(text) MOD maxNoTextLines) +j]
                  $(****Does not yet account for top and bottom margin)
              )
            ELSE
              VECTOR(FOR j FROM 1 TO maxNoTextLines:
                text[ (((i-LENGTH(output() )) MOD curLength) - 2)
                             * maxNoTextLines + j])
                  $(****Does not yet account for top and bottom margin)
            )  $("end of 'output() = VECTOR...")
          AND
            'spooledFiles() = spooledFiles() DIFF {cFileSeid}
          AND
            'cToTfile(cFile) = ?
          AND
            'copies(cFile) = ?
          AND
            textOfSeid(cToTfile(cFile)) = ?

          ;

$(The following OFUNS manipulate the VFUNS corresponding to ILPL events.)

OFUN OstartE();
  EFFECTS
    'VstartE() = FALSE;

OFUN OstopE();
  EFFECTS
    'VstopE() = FALSE;

OFUN OdieE();
  EFFECTS
    'VdieE() = FALSE;



END_MODULE
```

AD-A111 578   FORD AEROSPACE AND COMMUNICATIONS CORP PALO ALTO CA W--ETC F/G 9/2
KSOS COMPUTER PROGRAM DEVELOPMENT SPECIFICATIONS (TYPE B-5). (K--ETC(U)
DEC 80   MDA903-77-C-0333
UNCLASSIFIED   WDL-TR7932-REV-3   NL

4 OF 4
AD
A111578

END
DATE
FILMED
4-82
DTIC


1.0
4.5
5.0
5.6
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

```
MODULE enc        $(password encryption serivce)
MODULE enc        $(password encryption serivce)
$("
COPYRIGHT:  1978, by Ford Aerospace and Communications Corp.
ADDRESS:    Ford Aerospace and Communications Corp.
            Western Development Laboratories
            3939 Fabian Way
            Palo Alto, California 94303
            Attention: Software Technology Dept.
            Mail Stop: V02

CPCI:  NKSR
CPC:
MODULE NAME:  enc password encryption service
FUNCTION:  encryption of passwords
ASSUMPTIONS:  that the Govt. will give us an algorithm
HISTORY:
  AUTHOR:   Tom Berson
  VERSION:  1.1 of 10/18/78
  MODULE TYPE: SPECIAL Specifications
SPECIAL NOTES:
")


TYPES

password: VECTOR_OF CHAR ;
encryptedPw: {VECTOR_OF CHAR e|LENGTH(e)=ENCepwLength};


PARAMETERS

INTEGER ENCepwLength;


FUNCTIONS

VFUN ENCrypt(password pw)->encryptedPw epw:
 $(definition of the insides of this function to be provided by
  KSOS customer)


END_MODULE
```

ffn.specs   Page 1   Thu Apr 23 10.25:58 1981

```
MODULE ffn      $(file and terminal functions)
MODULE ffn      $(file and terminal functions)
$("
COPYRIGHT:  1978, by Ford Aerospace and Communications Corp.
ADDRESS:    Ford Aerospace and Communications Corp.
            Western Development Laboratories
            3939 Fabian Way
            Palo Alto, California 94303
            Attention: Software Technology Dept.
            Mail Stop: V02

CPCI:  NKSR
CPC:
MODULE NAME:  ffn file and terminal functions
FUNCTION:  sets the terminal level
ASSUMPTIONS:  none
HISTORY:
  AUTHOR:   Tom Berson
  VERSION:  1.1 of 10/18/78
  MODULE TYPE: SPECIAL Specifications
SPECIAL NOTES:  none
")


TYPES

terId: {0..255};
openDescriptor: {1..PSTmaxOpenDescriptors};
openModes: {omStatusUpdate};
accessLevel: STRUCT_OF(INTEGER securityLevel,integrityLevel;
                       SET_OF securityCat sCatSet);
tiiStruct: STRUCT_OF(INTEGER securityLevel; SET_OF securityCat sCatSet;
                     INTEGER integrityLevel; SET_OF integrityCat iCatSet;
                     INTEGER da; seid owner,group);


DECLARATIONS

openDescriptor odx,ody;


EXTERNALREFS

        FROM sen:
seid: DESIGNATOR;

        FROM smx:
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;

        FROM pst:
INTEGER PSTmaxOpenDescriptors;

        FROM ker:
VFUN K_get_object_level(INTEGER od)->tiiStruct level;
OFUN K_set_object_level(INTEGER od; tiiStruct level);
```

```
OVFUN K_open(seid oSeid; SET_OF openModes om;INTEGER stCap)
                ->openDescriptor od;

        FROM alc:
VFUN ALCsetPermissible(accessLevel proposed,maximum)->BOOLEAN b;

        FROM tdb:
VFUN TDBexists(terId tid)->BOOLEAN b;
VFUN TDBlocked(terId tid)->BOOLEAN b;
VFUN TDBstandardSeid(terId tid)->seid s;
VFUN TDBsecureSeid(terId tid)->seid s;
VFUN TDBmaxLevel(terId t)->accessLevel al;


FUNCTIONS

OFUN FFNsetTerminalLevel(terId tid;accessLevel al;
                seid newOwner, newGroup; INTEGER newDa);

        DEFINITIONS
 $(Note that in these definitions odx is not yet bound=>a scope
  error. However, odx is bound in the scope where the definitions
  are used.)
seid lowner IS IF newOwner=? THEN K_get_object_level(odx).owner
                              ELSE newOwner;
seid lgroup IS IF newGroup=? THEN K_get_object_level(odx).group
                              ELSE newGroup;
INTEGER lda IS IF newDa=? THEN K_get_object_level(odx).da
                           ELSE newDa;

        EXCEPTIONS
NEffnBadTerminal: ~TDBexists(tid);
NEffnTerminalLocked: TDBlocked(tid);
NEffnBadLevel: ~ALCsetPermissible(al,TDBmaxLevel(tid));

        EFFECTS
LET tiiStruct newLevel=STRUCT(al.securityLevel,al.sCatSet,
                              al.integrityLevel,?,
                              lda,lowner,lgroup)
    $(set the level of the standard path)
   IN LET odx=EFFECTS_OF K_open(TDBstandardSei[illegible]
                                {omStatusUpdate[illegible]
        IN EFFECTS_OF K_set_object_level(odx,newLevel)
           $(also set the level of the secure path)
           AND (LET ody=EFFECTS_OF K_open(TDBsecureSeid(tid),
                                          {omStatusUpdate},?)
                IN EFFECTS_OF K_set_object_level(ody,newLevel));


END_MODULE
```

```
MODULE gdb          $(group access authentication data base)
MODULE gdb          $(group access authentication data base)
$("
COPYRIGHT:  1978, by Ford Aerospace and Communications Corp.
ADDRESS:    Ford Aerospace and Communications Corp.
            Western Development Laboratories
            3939 Fabian Way
            Palo Alto, California 94303
            Attention: Software Technology Dept.
            Mail Stop: V02

CPCI:  NKSR
CPC:
MODULE NAME   gdb  group access authentication data ba
FUNCTION:  group access authentication data base

ASSUMPTIONS:  none
HISTORY:
  AUTHOR:   Tom Berson
  VERSION:  1.1 of 10/18/78
  MODULE TYPE: SPECIAL Specifications
SPECIAL NOTES:  none
")


TYPES

groupName: {VECTOR_OF CHAR g|LENGTH(g) >= 1};
encryptedPw: {VECTOR_OF CHAR e|LENGTH(e) = ENCepwLength};
userId:seid;
groupId: seid;
accessLevel: STRUCT_OF(INTEGER securityLevel,integrityLevel;
                         SET_OF securityCat sCatSet);


PARAMETERS

encryptedPw nullEpw;


EXTERNALREFS

        FROM sen:
seid: DESIGNATOR;

        FROM tii:
securityCat: DESIGNATOR;

        FROM enc:
INTEGER ENCepwLength;


ASSERTIONS

FORALL groupName x:
     CARDINALITY({groupId gid|GDBgroupName(gid) = x}) <= 1;
```

```
 $(In other words, we require unique group names.)


FUNCTIONS

 $(--------------- group access control info -------------)

VFUN GDBgroupName(groupId gid)->groupName gn;

        INITIALLY gn=?;


VFUN GDBencryptedPw(groupId gid)->encryptedPw e;
 $(group passwords are optional)

        INITIALLY e=nullEpw;


VFUN GDBadministrator(groupId gid)->userId admin;
 $(Only the group administrator may modify his group's password)

        INITIALLY admin=?;


VFUN GDBmaxLevel(groupId gid)->accessLevel l;

        INITIALLY l=?;


 $(------------ db extraction functions ------------)

VFUN GDBvalidGroupName(groupName gn)->BOOLEAN b;

        DERIVATION
EXISTS groupId gid : GDBgroupName(gid)=gn;


VFUN GDBgetGroupId(groupName gn)->groupId gid;

        EXCEPTIONS
NEgdbBadName. ~GDBvalidGroupName(gn);

        DERIVATION
SOME groupId x | GDBgroupName(x)=gn;

VFUN GDBpwExists(groupId gid)->BOOLEAN b;

        DERIVATION
NOT(GDBencryptedPw(gid) = nullEpw);


END_MODULE
```

```
MODULE pfn        $(process functions)
MODULE pfn        $(process functions)
$("
COPYRIGHT:  1978, by Ford Aerospace and Communications Corp.
ADDRESS:    Ford Aerospace and Communications Corp.
            Western Development Laboratories
            3939 Fabian Way
            Palo Alto, California 94303
            Attention: Software Technology Dept.
            Mail Stop: V02

CPCI:  NKSR
CPC:
MODULE NAME:  pfn  process functions
FUNCTION:  process functions:
        setting process level,
        getting process level,
        determination of whether we have an active shell at level L

ASSUMPTIONS:  none
HISTORY:
  AUTHOR:   Tom Berson
  VERSION:  1.1 of 10/18/78
  MODULE TYPE: SPECIAL Specifications
SPECIAL NOTES:  none
")


TYPES

userId: seid;
groupId: seid;
terId: {1..255};
accessLevel: STRUCT_OF(INTEGER securityLevel,integrityLevel;
                        SET_OF securityCat sCatSet);
tiiStruct:STRUCT_OF(INTEGER securityLevel;SET_OF securityCat sCatSet;
                    INTEGER integrityLevel;SET_OF integrityCat iCatSet;
                    INTEGER da; seid owner,group);
activeEntry:STRUCT_OF(terId tid;accessLevel al;seid pSeid);


PARAMETERS

seid PFNuserStarter;


EXTERNALREFS

        FROM sen:
seid: DESIGNATOR;

        FROM smx:
securityCat: DESIGNATOR;
integrityCat: DESIGNATOR;

        FROM ker:
```

```
VFUN K_get_process_level()->tiiStruct level;
OFUN K_set_process_level(tiiStruct level);
OVFUN K_spawn(seid pSeid;VECTOR_OF CHAR p1,p2)->seid newSeid;

        FROM udb:
VFUN UDBvalidUser(userId uid)->BOOLEAN b;
VFUN UDBinitialShell(userId uid)->VECTOR_OF CHAR s;
VFUN UDBloginDirectory(userId uid)->VECTOR_OF CHAR d;


FUNCTIONS

 $(--------- process level manipulation functions ----------)

OFUN PFNsetProcessLevel(accessLevel al; seid newOwner, newGroup;
                        INTEGER newDa);

        DEFINITIONS
seid lowner IS IF newOwner=? THEN K_get_process_level() owner
                              ELSE newOwner;
seid lgroup IS IF newGroup=? THEN K_get_process_level() group
                              ELSE newGroup;
INTEGER lda IS IF newDa=? THEN K_get_process_level().da
                           ELSE newDa;

        EFFECTS
LET tiiStruct old=K_get_process_level()
   IN LET tiiStruct new=STRUCT(
                al.securityLevel,al.sCatSet,
                al.integrityLevel,old.iCatSet,
                lda,lowner,lgroup)
         IN EFFECTS_OF K_set_process_level(new);

VFUN PFNgetProcessLevel()->STRUCT_OF(accessLevel al; userId owner;
                                     groupId group)pal;

        DERIVATION
LET tiiStruct s=K_get_process_level()
   IN STRUCT(STRUCT(s.securityLevel,s.integrityLevel,s.iCatSet),
             s.owner,s.group);


 $(------------- active shell level memory ------------)

VFUN h_activeSet()->SET_OF activeEntry aes:

        HIDDEN;
        INITIALLY aes=?;


OFUN PFNpostActive(activeEntry a);

        EXCEPTIONS
NEpfnActiveAlready: a INSET h_activeSet();

        EFFECTS
 ~h_activeSet()=h_activeSet() UNION {a};


OFUN PFNstrikeAllActiveShells(terId ptid);

        EFFECTS
 ~h_activeSet()=h_activeSet() DIFF
                {activeEntry ae|ae INSET h_activeSet()
                                AND ae.tid=ptid};


VFUN PFNqueryActive(terId ptid; accessLevel pal)->seid activeSeid;

        DERIVATION
IF EXISTS activeEntry ae:ae.tid=ptid AND ae.al=pal AND ae INSET h_activeSet()
          THEN ae.pSeid  $(not legal in SPECIAL, must be rebound)
          ELSE? ;  $(NB the ? here is a distinguished return)


 $(----------- common startup function ----------------)

OFUN PFNstartUser(terId tid; userId uid; accessLevel al );

        EXCEPTIONS
NEpfnBadUser: ~UDBvalidUser(uid);

        EFFECTS
LET seid startedSeid = EFFECTS_OF K_spawn(PFNuserStarter,
                           UDBinitialShell(uid),UDBloginDirectory(uid))
   IN EFFECTS_OF PFNpostActive(STRUCT(tid,al,startedSeid));


END_MODULE
```

```
MODULE sdb          $(system data base)
MODULE sdb          $(system data base)
$("
COPYRIGHT:  1978, by Ford Aerospace and Communications Corp.
ADDRESS:    Ford Aerospace and Communications Corp.
            Western Development Laboratories
            3939 Fabian Way
            Palo Alto, California 94303
            Attention: Software Technology Dept.
            Mail Stop: V02

CPCI:  NKSR
CPC:
MODULE NAME:  sdb  system data base
FUNCTION:  system level data base

ASSUMPTIONS:  none
HISTORY:
  AUTHOR:   Tom Berson
  VERSION:  1.1 of 10/18/78
  MODULE TYPE: SPECIAL Specifications
SPECIAL NOTES:  none
")


TYPES

accessLevel: STRUCT_OF(INTEGER securityLevel,integrityLevel;
                       SET_OF securityCat sCatSet);



EXTERNALREFS

        FROM smx:
securityCat: DESIGNATOR;


FUNCTIONS

VFUN SDBmaxLevel()->accessLevel l;

        INITIALLY l=?;


 $(and other functions required to support operator and administrator
  control functions.)
END_MODULE
```

```
MODULE tdb          $(terminal data base)

TYPES

terId:{0..255}:
accessLevel: STRUCT_OF(INTEGER securityLevel,integrityLevel;
                             SET_OF securityCat sCatSet):


PARAMETERS

 $(these contribute to the state of terminals between logout and login)
INTEGER TDBdefaultDa;
seid TDBdefaultOwner,TDBdefaultGroup;


EXTERNALREFS

        FROM sen:
seid: DESIGNATOR:

        FROM tii:
securityCat: DESIGNATOR;


FUNCTIONS

 $(---------------- terminal state information ------------)

VFUN TDBexists(terId tid)->BOOLEAN b;

        INITIALLY b=FALSE;

 $( terminals are brought into existence, i.e. configured, by the
   use of the system configuration editor. They arrive at reboot.)


VFUN TDBlocked(terId tid)->BOOLEAN b;

        INITIALLY b=FALSE;

 $( Terminals cannot be dynamically removed from configuration.
   They should be locked by the operator, and TDBexists then
   edited. They will then be gone at reboot. )


VFUN TDBstandardSeid(terId tid)->seid stdSeid;

        INITIALLY stdSeid=?;


VFUN TDBsecureSeid(terId tid)->seid secSeid;

        INITIALLY secSeid=?;


```

```
VFUN TDBmaxLevel(terId tid)->accessLevel l;

        INITIALLY l=?;

VFUN TDBdefaultSpeed(terId tid)->INTEGER speed;

        INITIALLY speed=?;


VFUN TDBdefaultKill(terId tid)->CHAR k;

        INITIALLY k=?;


VFUN TDBdefaultErase(terId tid)->CHAR e:

        INITIALLY e=?;


END_MODULE
```

```
MODULE uac      $(user access control functions)
 $( all functions in this module are subfunctions of the
   secure server, which is responsible for dialog with the
   user and for assembling the necessary parameters.)

TYPES

userId:seid;
groupId: seid;
terId:{1..255};
userName: {VECTOR_OF CHAR un | LENGTH(un) >= 1};
groupName: {VECTOR_OF CHAR gn|LENGTH(gn)>=1};
accessLevel: STRUCT_OF(INTEGER securityLevel,integrityLevel;
                       SET_OF securityCat sCatSet);
password: {VECTOR_OF CHAR n|LENGTH(n)>=1};
encryptedPw: {VECTOR_OF CHAR e|LENGTH(e)=ENCepwLength};


EXTERNALREFS

        FROM sen:
seid: DESIGNATOR;

        FROM tii:
securityCat: DESIGNATOR;

        FROM N_:
OFUN N_kill_all(seid terSeid);

        FROM alc:
VFUN ALCsetPermissible(accessLevel p,max)->BOOLEAN b;
accessLevel ALCgroundLevel;
accessLevel ALCloginLevel;

        FROM enc:
VFUN ENCrypt(password p)->encryptedPw epw;
INTEGER ENCepwLength;

        FROM sdb:
VFUN SDBmaxLevel()->accessLevel mal;

        FROM tdb:
userId TDBdefaultOwner;
groupId TDBdefaultGroup;
INTEGER TDBdefaultDa;
VFUN TDBlocked(terId tid)->BOOLEAN b;
VFUN TDBstandardSeid(terId tid)->seid stdSeid;
VFUN TDBmaxLevel(terId tid)->accessLevel mal;

        FROM udb:
VFUN UDBvalidUserName(userName un)->BOOLEAN b;
VFUN UDBgetUserId(userName un)->userId uid;
VFUN UDBencryptedPw(userId uid)->encryptedPw epw;
VFUN UDBdefaultGroup(userId uid)->groupId gid;
VFUN UDBmaxLevel(userId uid)->accessLevel mal;

```

```
        FROM gdb:
VFUN GDBvalidGroupName(groupName gn)->BOOLEAN b;
VFUN GDBgetGroupId(groupName gn)->groupId gid;
VFUN GDBpwExists(groupId gid)->BOOLEAN b;
VFUN GDBencryptedPw(groupId gid)->encryptedPw epw;
VFUN GDBmaxLevel(groupId gid)->accessLevel mal;

        FROM pfn:
VFUN PFNgetProcessLevel()->STRUCT_OF(accessLevel al;userId owner;
                                  groupId group)pal;
OFUN PFNsetProcessLevel(accessLevel al;userId newOwner;groupId newGroup;
                        INTEGER newDa);
OFUN PFNstartUser(terId tid; userId uid; accessLevel al);
VFUN PFNqueryActive(terId tid;accessLevel al)->seid pSeid;
OFUN PFNstrikeAllActiveShells(terId tid);

        FROM ffn:
OFUN FFNsetTerminalLevel(terId tid;accessLevel al;userId newOwner;
                         groupId newGroup;INTEGER newDa);
VFUN FFNgetTerminalLevel(terId tid)->STRUCT_OF(accessLevel al;
                        userId owner;groupId group;INTEGER da)tal;


FUNCTIONS

OFUN UAClogin(terId tid; userName un; password pw);

        DEFINITIONS
userId uid IS UDBgetUserId(un);

        EXCEPTIONS
NEuacTerLoggedIn: ~(FFNgetTerminalLevel(tid).owner=TDBdefaultOwner);
NEuacTerLocked: TDBlocked(tid);
NEuacBadLogin1: ~UDBvalidUserName(un);
NEuacBadLogin2: ~(UDBencryptedPw(uid)=ENCrypt(pw));

        EFFECTS
EFFECTS_OF FFNsetTerminalLevel(tid,ALCloginLevel,
                        uid,UDBdefaultGroup(uid),600);
EFFECTS_OF PFNsetProcessLevel(ALCloginLevel,uid,UDBdefaultGroup(uid),?);
EFFECTS_OF PFNstartUser(tid,uid,ALCloginLevel);





OFUN UAClogout(terId tid);

        EFFECTS
EFFECTS_OF N_kill_all(TDBstandardSeid(tid));
EFFECTS_OF PFNstrikeAllActiveShells(tid);
EFFECTS_OF FFNsetTerminalLevel(tid,ALCgroundLevel,TDBdefaultOwner,
                TDBdefaultGroup,TDBdefaultDa);
 $(EFFECTS_OF K_device_function to stty the terminal back to its
  default speed, kill, and erase characters)
EFFECTS_OF PFNsetProcessLevel(ALCgroundLevel,?,?,?);
```

```



OFUN UACchangeGroup(groupName gn; password pw):

        DEFINITIONS
groupId gid IS GDBgetGroupId(gn):

        EXCEPTIONS
NEuacBadGroup: ~GDBvalidGroupName(gn);
NEuacBadPw: ~(GDBpwExists(gid)=>GDBencryptedPw(gid)=ENCrypt(pw));
NEuacBalLevel: ~ALCsetPermissible(PFNgetProcessLevel().al,
                              GDBmaxLevel(gid));

        EFFECTS
EFFECTS_OF PFNsetProcessLevel(PFNgetProcessLevel().al,?,gid,?);


OFUN UACchangeAccessLevel(terId tid; accessLevel ral);

        DEFINITIONS
userId uid IS PFNgetProcessLevel().owner;
groupId gid IS PFNgetProcessLevel().group;

        EXCEPTIONS
NEuacBadLevel1: ~(ALCsetPermissible(ral,SDBmaxLevel()));
NEuacBadLevel2: ~(ALCsetPermissible(ral,TDBmaxLevel(tid)));
NEuacBadLevel3: ~(ALCsetPermissible(ral,UDBmaxLevel(uid)));
NEuacBadLevel4: ~(ALCsetPermissible(ral,GDBmaxLevel(gid)));

        EFFECTS
EFFECTS_OF PFNsetProcessLevel(ral,?,?,?);
EFFECTS_OF PFNsetTerminalLevel(tid,ral,?,?,?);
LET seid p=PFNqueryActive(tid,ral)
   IN IF(p=?$(OR p NOT active))THEN EFFECTS_OF PFNstartUser(tid,uid,ral)
                                ELSE TRUE;


END_MODULE
```

```
MODULE udb        $(user access authentication data base)

TYPES

userId:seid;
groupId:seid:
userName: {VECTOR_OF CHAR n|LENGTH(n)>=1};
encryptedPw: {VECTOR_OF CHAR e|LENGTH(e)=ENCepwLength}:
accessLevel: STRUCT_OF(INTEGER securityLevel,integrityLevel;
                        SET_OF securityCat sCatSet);


EXTERNALREFS

        FROM ser:
seid:DESIGNATOR;

        FROM tii:
securityCat:DESIGNATOR;

        FROM enc:
INTEGER ENCepwLength;


ASSERTIONS

FORALL userName x:
    CARDINALITY({userId uid|UDBloginName(uid)=x}) <= 1:
 $(In other words, unique user names are required)


FUNCTIONS

 $(----------- user access control state ---------------)

VFUN UDBloginName(userId uid)->userName n;

        INITIALLY n=?;


VFUN UDBencryptedPassword(userId uid)->encryptedPw p:

        INITIALLY p=?;


VFUN UDBdefaultGroup(userId uid)->groupId gid:

        INITIALLY gid=?;


VFUN UDBloginDirectory(userId uid)->VECTOR_OF CHAR dName;

        INITIALLY dName=?;


VFUN UBDinitialShell(userId uid)->VECTOR_OF CHAR sName;
```

```

        INITIALLY sName=?;


VFUN UDBmaxLevel(userId uid)->accessLevel l;

        INITIALLY l=?;


 $(------------- db extraction functions -------------)

VFUN UDBvalidUserName(userName un)->BOOLEAN b;

        DERIVATION
EXISTS userId x:UDBloginName(x)=un;


VFUN UDBgetUserId(userName un)->userId uid;

        EXCEPTIONS
NEudbBadName: ~UDBvalidUserName(un);

        DERIVATION
SOME userId x|UDBloginName(x) = un;


END_MODULE
```

```

$("      MODULE           udm.specs (Version 1.5,14:55:05)
                          Module Name (udm.specs)
                          File Name (/kr/sunix/specs/nksr/s.udm.specs)
         CONTENTS         directory manager
         TYPE             SPECIAL.specifications
         LAST CHANGED:    11/21/80,14:55:05

")

MODULE udm

      TYPES
dirName:  VECTOR_OF CHAR:
path: VECTOR_OF dirName;
nonDisType: STRUCT_OF(
                 INTEGER securityLevel: SET_OF securityCat securityCatS;
                 INTEGER integrityLevel; SET_OF integrityCat integrityCatS):
  $(integrityCat is typically the null set)
daType: SET_OF daMode:
modeStruct: STRUCT_OF(daType ownerMode, groupMode, allMode);
tiiStruct: STRUCT_OF(nonDisType nd;
                 modeStruct da; INTEGER owner, group; SET_OF privType priv);


      PARAMETERS
seid  ROOTseid $(root of the directory system),
       sameNspSeid, $(special seid whose nsp component is "sameNsp"
                      to indicate whether or not a file or directory x
                      has been mounted.  x has been mounted iff
                      its nsp component is not "sameNsp".)
       zeroFive,  $(special seid whose index field corresponds
                  to the implementation value <0,5> which represents
                  mounted files or directories.  This parameter is  used
                  by the VFUN derivedDirectory.)
       slashTmpSeid,      $("The seid of the directory /tmp.
                          This seid could be defined as
                          simpPathInterpret(pSeid,ROOTseid,<"tmp">),
                          where simpPathInterpret is obtained from
                          pathInterpret (below) by using directory
                          instead of derivedDirectory")
       usersTmpSeid;      $(" The seid of the user's unique tmp directory,
                          which exists under /tmp.  usersTmpSeid could be
                          defined as simpPathInterpret(pSeid,ROOTseid,
                          <"tmp","uniqueName">), where simpPathInterpret
                          is obtained from pathInterpret by using
                          directory instead of derivedDirectory.")


      DEFINITIONS
BOOLEAN mountEquiv(seid  s1,s2) IS
  s1 = s2 OR
  (SENseidIndex(s1) = SENseidIndex(s2)
  AND
  SENseidNsp(s1)  = SENseidNsp(sameNspSeid));
  $("True iff s2 = s1 or s2 has been mounted on s1.  mountEquiv
  is used in existsPath (below).  Basic idea is that IF
  a sequence of unmounts were to be done, then there
  would be a "direct" path in the definition of existsPath.")

BOOLEAN existsPath(seid start,stop) IS
   (EXISTS INTEGER n | n > 1:
     (EXISTS VECTOR_OF dirName d | LENGTH(d) = n-1
                       AND (FORALL INTEGER j|  j INSET {1 .. n-1} :
                       NOT d[j] INSET {".", ".."}) :
        (EXISTS VECTOR_OF seid s | (LENGTH(s) = n
                                   AND s[1] = start
                                   AND s[n] = stop) :
           (FORALL INTEGER i| i INSET {1 .. n-1} :
                                    mountEquiv(directory(s[i],d[i]),s[i+1]))
    )));

seid realDir(seid dirSeid; dirName name) IS
    IF dirSeid = usersTmpSeid AND name = ".." THEN
      slashTmpSeid
    ELSE IF dirSeid = slashTmpSeid AND NOT name INSET {".",".."} THEN
      usersTmpSeid
    ELSE
      dirSeid;

seid pathInterpret(seid pSeid,dirSeid; path p)
  $("This definition, in conjunction with the auxilliary one pil
    given below, essentially tries to work its  way down a path
    p, starting at node dirSeid,
    a node at a time, returning as a value the last node
    encountered.  Various exceptions must be met  successfully
    before moving from one node to the next.  The notion of a path
    as used in these specs as an abstraction of the standard Unix
    notion of a path.  For example, the path /a/b/c  would be
    represented simply as the sequence of names a b c.
  ")
  IS
   IF isDirectory(dirSeid) THEN
       $(Starting node must be directory to continue down path)
     IF         (
                 SMXdap(pSeid,dirSeid,{daRead})
                OR
                 SMXdap(pSeid,dirSeid,{daExecute})
                )
                AND
                 SMXflow(pSeid,dirSeid,{daRead})
              $(Must have discretionary read or execute permission,
                and must have mandatory  read flow to continue down path)
     THEN
       IF LENGTH(p) > 0 THEN
         pil(pSeid,dirSeid,p)   $(Start recursive call to move
                                    down the path)
       ELSE  $(path is of length 0, so return dirSeid)
         dirSeid
     ELSE  $(flow violated)
       ?
   ELSE $(starting node is not a directory)
```

```
    ?
  :


seid pil(seid pSeid,dirSeid; path p)
  $(An auxilliary definition  used by pathInterpret)
  $(We know at the beginning of each recursive call that
    dirSeid is a directory to which pSeid has access, and that
    path p is non-null.)
 IS
  IF derivedDirectory(dirSeid,p[1]) ~= ?  THEN
     $(There must be a well-defined
      next node along the path.)
    IF            (
                    SMXdap(pSeid,derivedDirectory(dirSeid,p[1]), {daRead})
                  OR
                    SMXdap(pSeid,derivedDirectory(dirSeid.p[1]), {daExecute})
                  )
                AND
                   SMXflow(pSeid,derivedDirectory(dirSeid.p[1]),{daRead})
               $(Must have discretionary read or execute permission,
                 and must have mandatory  read flow to continue down path)
    THEN
      IF LENGTH(p) > 1 THEN
        IF isDirectory(derivedDirectory(dirSeid,p[1])) THEN
                 $(basic recursion:  move one node down the path)
          pil(pSeid,derivedDirectory(dirSeid,p[1]),
                           VECTOR(FOR i FROM 2 TO LENGTH(p): p[i]))
        ELSE  $(There are at least two more nodes along the path,
              but the next one is not a directory, so return ?)
          ?
      ELSE $(The next node is the last one in the path, so return it)
        derivedDirectory(dirSeid,p[1])
    ELSE  $(Flow or dap violations encountered)
      ?
  ELSE  $(There is not a well-defined next node along the path)
    ?
  ;


     EXTERNALREFS

    FROM smx:
seid: DESIGNATOR;
secureEntityType: {tFile, tDevice, tTerminal, tProcess, tSegment, tSubtype,
                   tExtent, tNull};
privType: {
        privFileUpdateStatus,   privLink,       privLockSeg,
        privModifyPriv,         privMount,
        privSetFileLevel,       privSetSegProcLevel,
        privStickySeg,          privTerminalLock,
        privViolSimpSecurity,   privViolStarSecurity,
        privViolSimpIntegrity,  privViolStarIntegrity,
        privViolDiscrAccess,    privSignal,     privWalkPTable,
        privHalt,               privKernelCall, privViolCompartments,
        privRealizeExecPermissions};
```

```

daMode: {daRead, daWrite, daExecute};
securityCat: DESIGNATOR:
integrityCat: DESIGNATOR:
domainType: {userDomain, supervisorDomain};

VFUN TIIinfo(seid s) -> tiiStruct st;
VFUN SMXflow(seid pSeid, oSeid; daType da) -> BOOLEAN b·                $(SMXflow)
VFUN SMXdap(seid pSeid, oSeid; daType da) -> BOOLEAN b;                  $(SMXdap)
VFUN SENseidNsp(seid anySeid) -> INTEGER nsp:                        $(SENseidNsp)
VFUN SENseidIndex(seid anySeid) -> INTEGER index;
VFUN SENmakeSeid(seid exampleSeid; INTEGER index) -> seid rSeid;
VFUN SENseidType(seid s) -> secureEntityType set;


     ASSERTIONS
$(The following assertions deal with the basic tree structue
of directories.)

$(ASSERTION 1)
FORALL seid s | isDirectory(s): directory(s,".") = s;
$(Each directory links to itself via ".")

$(ASSERTION 2)
FORALL seid parentSeid; seid childSeid |
    (isDirectory(parentSeid)
    AND isDirectory(childSeid)
    AND  parent(childSeid) = parentSeid):
       ((EXISTS dirName name: directory(parentSeid,name)
                = childSeid)
        AND directory(childSeid,"..") = parentSeid);
$(Whenever the ghost variable parent points from child to parent,
there is a real link from parent to child via some name, and there
is a real link from child to parent via "..")


$(ASSERTION 3)
FORALL seid s | isDirectory(s): existsPath(ROOTseid,s);
$(Every directory is reachable starting from the root.)
$(The only link to the root is via the root itself.)

$(ASSERTION 4)
FORALL seid s1:seid s2 |
        (isDirectory(s1) AND isDirectory(s2) AND s1~= s2):
        (NOT (existsPath(s1,s2) AND existsPath(s2,s1)))·
$(No cycles exist between distinct nodes.)



     FUNCTIONS

$(-----------------VFUNS-----------------)

VFUN directory(seid dirSeid; dirName name) -> seid ansSeid;
        HIDDEN;
        INITIALLY
        ansSeid =
        (IF (dirSeid = ROOTseid AND name INSET {".", ".."}) THEN
            ROOTseid
        ELSE
            ?);
$(Initialized by mkDir; updated by mkEntry,rwEntry; finalized by remove.)

VFUN derivedDirectory(seid dirSeid; dirName name) -> seid ansSeid;
  $(For certain functions, such as path interpret, derivedDirectory
    rather than directory is utilized.  directory corresponds to
    the implementation-visible table, whereas derivedDirectory
    returns a run-time computable value based on this table.)
  DEFINITIONS

  seid s IS directory(realDir(dirSeid,name),name);

  EXCEPTIONS
    badLogic: FALSE;  $(Placeholder for exceptions)
  DERIVATION
   (IF s ~=? THEN
      IF SENseidNsp(s) = SENseidNsp(sameNspSeid) THEN
        SENmakeSeid(realDir(dirSeid,name),SENseidIndex(s))
      ELSE IF SENseidType(s) = tFile THEN  $(mount processing)
        SENmakeSeid(s,SENseidIndex(zeroFive))
      ELSE  $(non-file type  in directory.  This had better be
                         the last component of a path)
        s
    ELSE
      ?
    );


VFUN isDirectory(seid dirSeid) -> BOOLEAN b:
        HIDDEN;
        INITIALLY
        b = (dirSeid = ROOTseid);
$(Initialized by mkDir; finalized by remove.)

VFUN parent(seid dirSeid) -> seid s;
        HIDDEN;
        INITIALLY
        s = (IF dirSeid = ROOTseid THEN ROOTseid ELSE ?);
$(Ghost variable corresponding to the double-dot entry in each
directory.  Initialized by mkDir; finalized by remove.)


$(-------------------OFUNS------------------------)

OFUN mkEntry(seid pdirSeid, eSeid; dirName name1)[seid pSeid];
  $("Link from (dirSeid,name1) to eSeid.  eSeid must not be
  a directory - if it is, the OFUN mkDir is used instead.")

        DEFINITIONS
  seid dirSeid IS realDir(pdirSeid,name1);

        EXCEPTIONS
  badFlow: NOT isDirectory(dirSeid) OR NOT SMXflow(pSeid,dirSeid,
               {daRead,daWrite});
  badDap:  NOT SMXdap(pSeid,dirSeid,{daExecute,daWrite});
  NotSameFileSystem:  SENseidNsp(dirSeid) ~= SENseidNsp(eSeid);
  badarg: isDirectory(eSeid);
  nameExists:  directory(dirSeid,name1)~=  ?;
  RESOURCE_ERROR;

         EFFECTS
  'directory(dirSeid,name1) = eSeid;


OFUN mount(seid pdirSeid, nspSeid: dirName name) [seid pSeid,parentpSeid];
  $("mount a new extent on directory(dirSeid,name), by replacing
    its nsp field with that of nspSeid.")
  $(The object being mounted can only be a certified KSOS file system;
   this is enforced by the mount CPC)
  $("The additional implicit parameter parentpSeid is the seid
  of the parent process of pSeid.  It is derivable from pSeid by
  bringing up additional state information from the kernel specs,
  but for simplicity here is given in this form.  Its use here
  is in the notMountpriv exception below")

       DEFINITIONS
  seid dirSeid IS realDir(pdirSeid, name);
  seid d IS directory(dirSeid,name) ;

       EXCEPTIONS
  badFlow: NOT isDirectory(dirSeid) OR NOT SMXflow(pSeid,dirSeid,
               {daRead,daWrite});
  badDap:  NOT SMXdap(pSeid,dirSeid,{daRead,daWrite});
  badName: directory(dirSeid,name) = ?;
  notMountpriv:  NOT privMount INSET TIIinfo(parentpSeid).priv;
  cannotMountHere:  SENseidNsp(sameNspSeid) ~= SENseidNsp(d) ;
  notFileType:  SENseidType(d) ~=tFile; $(The object upon which we are
                               mounting must be a file, which includes
                               directories)

        EFFECTS
  'directory(dirSeid,name) = SENmakeSeid(nspSeid,
               SENseidIndex(directory(dirSeid,name)));

OFUN unmount(seid pdirSeid: dirName name) [seid pSeid,parentpSeid];
  $("Undo the mount operation by restoring the nsp field of
    directory(dirSeid,name) to the value sameNsp.")
  $("The additional implicit parameter parentpSeid is the seid
  of the parent process of pSeid.  It is derivable from pSeid by
  bringing up additional state information from the kernel specs,
  but for simplicity here is given in this form.  Its use here
  is in the notMountpriv exception below")

       DEFINITIONS
  seid dirSeid IS realDir(dirSeid,name);

       EXCEPTIONS
  badFlow: NOT isDirectory(dirSeid) OR NOT SMXflow(pSeid,dirSeid,
                     {daRead,daWrite});
    badDap:  NOT SMXdap(pSeid,dirSeid,{daRead,daWrite});
    badName: directory(dirSeid,name) = ?;
    notMountpriv:  NOT privMount INSET TIIinfo(parentpSeid).priv;

          EFFECTS
    'directory(dirSeid,name) = SENmakeSeid(sameNspSeid,
                     SENseidIndex(directory(dirSeid,name)));


OFUN mkDir(seid pparentSeid; dirName name1) [seid pSeid];
    $(Create a new directory as a child of parentSeid, linking from
    parent to child via name1.  The child seid is defined below.
    Initialize the new directory with dot and double-dot entries, and
    maintain invariance of tree assertions.)

          DEFINITIONS
    seid parentSeid IS realDir(pparentSeid,name1);
    seid childSeid IS SOME seid s |
          TIIinfo(s) =? AND SENseidNsp(s) = SENseidNsp(parentSeid);

          EXCEPTIONS
    badFlow: NOT isDirectory(parentSeid) OR NOT SMXflow(pSeid,parentSeid,
                     {daRead,daWrite});
    badDap:  NOT SMXdap(pSeid,parentSeid,{daExecute,daWrite});
    badSeid: childSeid =?;
    nameExists: directory(parentSeid,name1) ~= ?;

          EFFECTS
    'directory(parentSeid,name1)= childSeid;
    'TIIinfo(childSeid)=  TIIinfo(pSeid);
    'directory(childSeid,".")= childSeid;
    'directory(childSeid,"..")= parentSeid;
    'parent(childSeid) = parentSeid;
    'isDirectory(childSeid) = TRUE;


OFUN remove(seid pparentSeid; dirName name1)[seid pSeid];
    $("Unlink the (parentSeid, name1) entry.  If the child is a directory,
    it must be empty, and pass da checks.")

          DEFINITIONS
    seid parentSeid IS realDir(pparentSeid,name1);
    seid aSeid IS directory(parentSeid,name1);

          EXCEPTIONS
    badFlow: NOT isDirectory(parentSeid) OR NOT SMXflow(pSeid,parentSeid,
                     {daRead,daWrite});
    badDap:  NOT SMXdap(pSeid,parentSeid,{daExecute,daWrite});
    badName:  directory(parentSeid,name1)=?;
    badDap2:  isDirectory(aSeid) AND NOT SMXdap(pSeid,aSeid,{daRead,daWrite});
    notEmpty:  isDirectory(aSeid) AND (EXISTS dirName name2 |
                     NOT name2 INSET  { ".", ".."};
                     directory(aSeid,name2)~=?);

          EFFECTS
```

```
  ~directory(parentSeid,name1) = ?;
  isDirectory(aSeid) => ~directory(aSeid,".")= ?
                    AND ~directory(aSeid,"..") = ?
                    AND ~isDirectory(aSeid) = FALSE
                    AND ~parent(aSeid) = ?;

$(The following OFUNs, piRemove,piMkEntry, piLocate, and piChangeDir,
are provided for the KSOS emulator.  Their purpose here is simply
to let the caller know whether an exception would be [illegible] by
a call to remove, mkEntry, locate, and changeDir respectively.
Thus, in the OFUNs below, the EFFECTS sectio
```

DATE
FILMED

8